Clube Delphi

Ano 12 :: Edição 140 :: R$ 11,90

DevMedia

Gerência de banco de dados
Conheça os recursos do RapidSQL XE 2

Minicurso
NFe no Delphi XE 2 – Parte 3
Consumindo os WebServices da SEFAZ

DELPHI E OO

Domine para fazer a diferença

Silverlight 5 no Prism XE 2
As novidades desta versão, na prática

Boas Práticas – Refatoração
Técnicas para melhorar seu código

ISSN 1517990-7

Somos tão apaixonados por tecnologia que o nome da empresa

diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,

assim como você.

Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única. Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486

Sumário

Delphi

Novidades

08 – Silverlight 5 com Prism XE 2
[*Fabrício Hissao Kawata*]

Banco de Dados

16 – Gerenciando banco de dados
[*Fabrício Hissao Kawata*]

Minicurso

24 – NF2 no Delphi XE 2 – Parte 3
[*Giuliano Scombatti Pinto*]

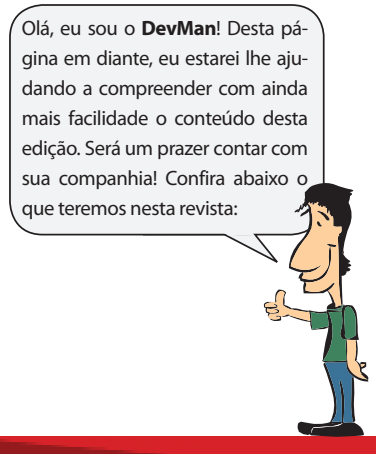
Boas Práticas

Boas Práticas

32 – Técnicas avançadas de POO
[*Giuliano Scombatti Pinto*]

Boas Práticas

44 – Refatoração Fácil
[*Giuliano Scombatti Pinto*]



Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Clube Delphi

Ano 12 - 140ª Edição 2012 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral

Paulo Quicoli
pauloquicoli@gmail.com

Editor Técnico

Daniel Sobrinho Laporte
daniel.laporte@gmail.com

Equipe Editorial

Guinther Pauli
guintherpauli@gmail.com

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br

Giuliano Scombatti Pinto
giuliano@sygnux.com.br

Jornalista Responsável

Kaline Dolabella - JP24185

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/mancad
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

EDITORIAL

Programar de forma orientada a objetos é o desejo de muito, porém a falta de conhecimento e de tempo para aprender a forma correta, as boas práticas, acabam impedindo isso de acontecer. Mas temos nesta edição um artigo do Guinther Pauli que poderá trazer seus sonhos à realidade. Explore as melhores práticas da programação OO com exemplos que são fáceis de entender e então aplicar em sua realidade.

Ainda falando de orientação a objetos, temos um artigo do Giuliano que explora os tipos de refatoração existentes. Aplicar refactoring é algo de que muitos desenvolvedores fogem pois sempre está relacionado a muito trabalho. Contudo, o IDE do Delphi XE 2 oferece recursos que facilitam sua aplicação. Não perca.

A tecnologia evolui, sua concorrência também. Vários e diversos bancos de dados surgiram e estão aí prontos para o uso. Porém, temos aqueles que se destacam: SQL Server, Oracle, Firebird, PostgreSQL e outros. E sua empresa para não perder concorrência, precisa estar pronta para utilizar todos estes, assim, também gerenciá-los. Melhor do que ter várias ferramentas para cada um deles, é ter uma só, que ainda te ajude na administração e até mesmo na criação de scripts SQL. Este é objetivo do RapidSQL XE 2 que o Fabrício apresenta em seu artigo.

Apesar do plugin Flash estar deixando de ser utilizado, existe outro focado no ambiente Windows, destinado às intrawebs empresariais, o Silverlight. Desenvolvido pela Microsoft, este é utilizado para o desenvolvimento de aplicativos comerciais para Web, aplicando o conceito de melhoria na interface do usuário. O desenvolver Delphi também pode utilizar essa tecnologia através do Prism XE 2 e agora, tirar proveito de todos os novos recursos trazidos pela versão 5 do Silverlight. Não sabe quais são esses recursos? Leia o artigo do Fabrício.

E continuando ainda com o assunto Nota Fiscal Eletrônica, o Giuliano mostra como utilizar os WebServices da SEFAZ e conceitos por traz dessa utilização.

Boa leitura!

Paulo Quicoli
pauloquicoli@gmail.com
Editor Geral
quicoli.wordpress.com
twitter.com/pauloquicoli

MVP

Assine agora e tenha acesso a todo o conteúdo da DevMedia:
www.devmedia.com.br/mvp

DevMedia

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Paulo Quicoli - Editor da Revista
pauloquicoli@gmail.com

TENHA ACESSO ILIMITADO A TODO CONTEÚDO DO SITE DEVMEDIA

SEJA UM ASSINANTE MVP!

Tenha acesso garantido a qualquer tema que você possa necessitar!

VIDEOS -AULAS

O assinante MVP acessa todas as vídeo-aulas do site DevMedia. Nenhum outro produto dá essa possibilidade. São mais de **7.000 video-aulas.**

CURSOS

Nosso site possui atualmente **+ de 250 cursos online** com a qualidade que você já conhece e confia. Com o MVP você terá acesso a todos eles.

REVISTAS

Assinante MVP acessa todas as nossas revistas na versão digital, incluindo todas as edições anteriores. Quando pintar aquela dúvida você terá mais de **3.500 artigos para consultar!**

ASSINATURA MVP MOST VALUABLE PROFESSIONAL

A assinatura MVP te dá a oportunidade de aumentar seu conhecimento em diversos assuntos, com qualidade e economia.

Invista na sua carreira se torne um profissional ainda mais completo!

POR APENAS R$59,90 POR MÊS

DevMedia

Acesse: www.devmedia.com.br/mvp

A evolução das Interfaces c

Desde o surgimento do Windows, várias evoluções ocorreram, principalmente com a introdução da GUI. Porém, embora várias evoluções sejam facilmente perceptíveis, uma em específico tornou-se um padrão cada vez mais colocado em prática e aperfeiçoado mesmo nos dias de hoje, o Drag and Drop, ou arrastar e soltar. Logo no início dos Sistemas Operacionais com interfaces gráficas era possível deslocar facilmente arquivos para uma estrutura de arquivos (pasta), mover arquivos de uma posição para outra (coordenadas X e Y), ou mesmo de uma estrutura de pastas para dentro de outra, tudo isso é realizado pelo recurso de Drag and Drop.

Hoje em dia é possível notar a utilização deste recurso de uma forma diferente. Tudo começou com a introdução a um novo conceito de arrastar e soltar, até então, mostrado em filmes de ficção científica. A Microsoft introduziu através da tecnologia Microsoft Surface um novo modelo de arrastar e soltar, na qual o usuário interagia com os elementos da tela, movendo-os de um lado para o outro da superfície, definindo sua disposição, tamanho e etc. A inovação deu tão certo que foi parar em Smartphones e hoje se encontra até mesmo em Sistemas Operacionais.

Realizando um overview de um passado não muito distante ao presente atual, é possível notar desde a era do antigo Windows Mobile o aperfeiçoamento dos Smartphones, fazendo com que o usuário pudesse ter o controle sobre os aplicativos ou até mesmo do Sistema Operacional por meio do tato. Hoje em dia, o crescimento mostra-se ainda maior com a chegada do iOS para o iPhone, Windows Phone 7 e Android. É claro que estes recursos são derivados das tecnologias baseadas em Touch / Multi-Touch que diferem dos recursos de Drag and Drop, porém, deve-se lembrar que desde antes já se controlava disposição de objetos projetados em uma tela através da técnica de coordenadas. Esta tecnologia de Touch foi inserida logo no Windows 7, porém, tornou-se ainda mais presente atualmente na nova versão do Sistemas Operacional da Microsoft, o Windows 8, na qual assemelha-se muito de aparência com o Windows Phone 7, possuindo um menu Iniciar totalmente diferenciado e personalizável.

Voltando o foco para o Drag and Drop, este recurso combina muito bem com outra funcionalidade presente no Delphi, a RTTI. A RTTI (RunTime Type Information) é um recurso extremamente poderoso e amplamente utilizando na orientação a objetos. Através dela, é possível obter praticamente qualquer tipo de informação de forma dinâmica. É possível ver uma utilização abrangente da RTTI em Frameworks de ORM (mapeamento objeto-relacional), onde é possível obter todas as informações de um objeto, como propriedades, métodos entre muitas outras coisas. Outro exemplo prático que ilustra o uso da RTTI com orientação a objetos é a utilização de serialização de objetos por exemplo, o que é algo comum nas versões novas do Delphi por meio de JSON.

Embora todos estes conceitos mencionados utilizem RTTI, um em específico chama a atenção por está presente no cotidiano o tempo todo, dentro do próprio IDE do Delphi. Ao abrir o IDE do Delphi logo estão presentes os recursos de Drag and Drop e RTTI através da Tool Palete e do Object Inspector. Isto fica mais perceptível ao criar um novo formulário, clicar na Tool Palete e arrastar um componente para o formulário. Ao clicar sobre o componente, ou formulário, suas informações são automaticamente mapeadas e exibidas no Object Inspector. Isso tudo é realizado de forma transparente, sem que possa ser notado pelo desenvolvedor no primeiro momento, porém, esses recursos nada mais são do que RTTI e Drag and Drop.

Criando um pequeno exemplo de Drag and Drop

Um pequeno exemplo de Drag and Drop ilustra como lidar com situações de arrastar e soltar para diferentes tipos de componentes. Aqui são utilizados um Button e um Edit. Para trabalhar com o Drag and Drop é importante saber quem será o objeto origem (a ser arrastado) e qual será seu destino (outro objeto). O exemplo ilustra como origem os componentes Button e Edit, permitindo movê-los dentro da área do formulário, origem neste caso (**Figura 1**).

Nota do DevMan

O Windows Mobile recentemente extinto, é um Sistema Operacional para dispositivos móveis, geralmente utilizado em Smartphones, celulares e dispositivos multimídia. Utilizava como base o Compact .net Framework 3.5 para o desenvolvimento de suas aplicações por meio de um SDK e do IDE Visual Studio 2008. Foi descontinuado na versão 6.5 e substituída pelo até então Windows Phone 7, baseado na tecnologia Silverlight.

Nota do DevMan

GUI é a sigla de Graphical User Inteface ou Interface gráfica do Usuário. Basicamente é onde se deu início a um novo modelo de interface com maior interatividade através de elementos gráficos como ícones por exemplo. Tornou-se de fato um produto graças a Apple, pois até então, tudo era realizado por interfaces de comando como o prompt de comando (MS-DOS).

Nota

Os recursos de Touch e Gestures foram suportados pelo Delphi a partir da versão 2009. São amplamente utilizados em caixa eletrônicos, terminais de venda, drive thru e etc. A partir do Windows 7 o suporte a estes recursos também foi introduzido, desde que se possua o hardware suportado.

om conceitos já existentes

Daniel Sobrinho Laporte

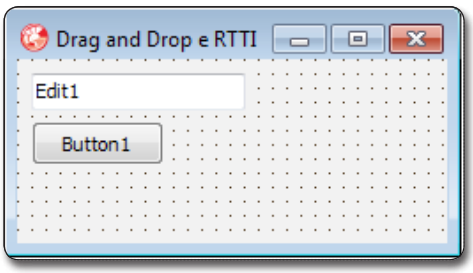


Figura 1. Formulário de Drag and Drop

Para ambos os componentes, o evento utilizado nos componentes de origem é o OnMouseDown, que ocorre de acordo com as coordenadas do mouse e que no caso estão compartilhados. A RTTI é utilizada para verificar se o Sender (origem) é um tipo TButton ou TEdit, iniciando o método BeginDrag, que irá notificar que é possível deslocar o objeto.

```
procedure TForm1.Edit1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Sender is TButton then
    TButton(Sender).BeginDrag(False)
  else
    TEdit(Sender).BeginDrag(False);
end;
```

Por último o evento de destino, que é codificado dentro do próprio formulário. O evento Drag Over ocorre quando é o fim do movimento de arrastar. É possível notar os parâmetros x e y (coordenadas), assim como Sender e Source.

```
procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
  Accept := (Source is TEdit) or (Source is TButton);
  if Accept then
  begin
    if Source is TButton then
    begin
      TButton(Source).Left := x;
      TButton(Source).Top  := y;
    end
    else if Source is TEdit then
    begin
      TEdit(Source).Left := x;
      TEdit(Source).Top  := y;
    end;
  end;
end;
```

Nota-se que a variável Accept é quem controla o final da execução do evento, ocorrendo quando for identificado como destino o componente TEdit ou TButton. A seguir, o componente recebido como parâmetro apenas trata de se deslocar para as coordenadas recebidas e que referem-se a posição do cursor do mouse.

Daniel Sobrinho Laporte
daniel.laporte@gmail.com / www.twitter.com/daniel_laporte
Colunista e editor técnico da Revista ClubeDelphi. Graduado em Sistemas de informação e pós-graduado em Administração de Bancos de Dados. É analista de sistemas e desenvolvedor Delphi desde 2005 desenvolvendo aplicações cliente/servidor e multicamadas.

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Silverlight 5 no Prism

Uma abordagem sobre as novidades da

Basicamente, o Silverlight pode ser definido como o Framework da Microsoft voltado para a construção de aplicações ricas de internet, que poderão estar alocadas em um navegador Web ou não, e com a possibilidade de serem executadas em qualquer Sistema Operacional que suporte a tecnologia. No cenário da Web, este tipo de aplicação denominada "rica", se mostra muito mais poderosa e atraente, se comparado a uma página tradicional HTML, por exemplo. Um dos principais motivos a serem citados para a justificativa é a sua proximidade com as aplicações Desktop em termos de usabilidade, além de um ganho ainda maior na experiência do usuário final. Soma-se a isso seus recursos gráficos acelerados por hardware, recursos 2D e 3D e animações vetoriais.

Ainda sobre sua execução, o Framework utiliza-se de um pequeno item que é o responsável por tornar o conteúdo Silverlight disponível para uma página no Browser ou uma aplicação independente (out-of-browser). O item em questão trata-se de um simples plug-in, que é o único elemento necessário a ser instalado pelo usuário final em sua máquina. Atualmente (até o momento da escrita deste artigo) o Silverlight se encontra em sua versão estável de número 5. Apesar de sua premissa inicial ter reforçado o seu conceito cross-plataform, a quantidade de plataformas suportadas ainda é razoável, sendo as principais Windows e Mac.

Resumo DevMan

De que trata o artigo:

O Microsoft Silverlight é um Framework que nasceu sob o nome de WPF/E, um abreviação de Windows Presentation Foundation for Everyone (algo como WPF para todos). Neste momento ainda tratava-se apenas de um mero subconjunto de classes e recursos do WPF, tecnologia esta que era um dos pilares do .NET Framework 3.0 e destinado a construção de aplicações de interfaces de usuário ricas. Pouco tempo depois, já batizado efetivamente de Silverlight, este tomou rumos próprios, ainda herdando sua base do WPF, mas acrescentando importantes recursos próprios. Sua mais recente versão, de número 5, entra em cena com muitas melhorias que culminam em novas possibilidades para os desenvolvedores. O presente artigo faz então uma apresentação inicial de algumas destas novidades, deixando o leitor bem situado no nível de evolução atingido pelo Framework. Tudo isto disponível ao mundo Delphi, através do Embarcadero Prism XE2.

Em que situação o tema é útil:

No cenário atual, é fácil perceber que o Silverlight não é apenas mais uma tecnologia para se criar páginas simples e aplicações Web comuns, de interface rica e poderoso poder de interatividade. Ele é atribuído hoje como sendo uma boa solução para a linha de aplicações comerciais, mais precisamente para a construção de LOB (Line-of-Business) Applications. Dessa forma, o profissional Delphi que queira desenvolver aplicações deste segmento, que possuem uma UI (User Interface) mais rica e um alto grau de experiência do usuário, pode adotar o Silverlight. Partindo desta escolha, nada melhor que se manter inteirado do estágio atual em que se encontra o Framework, que é justamente a premissa principal do artigo.

Silverlight 5 no Prism XE2:

O Silverlight foi concebido como sendo a principal solução da Microsoft para a construção de aplicações e páginas ricas para Web, já sendo interpretado de imediato como sendo um concorrente direto do Adobe Flex, até então dominante no mercado. No entanto, de lá pra cá, o que se vê é um cenário totalmente diferente até mesmo das previsões mais radicais. O Flash, quase único anteriormente, começa a perder terreno para outras tecnologias, tais como o novo e badalado HTML 5. Prova desta perda é o lançamento de novos produtos (iPhone e Windows Phone, por exemplo) que já não oferecem suporte a ele. Na linha oposta, o próprio Silverlight, dito como seu principal concorrente, agora se estabelece cada vez mais em outro contexto, que é o das aplicações de negócios (Line-of-Business Applications) e, mais recentemente tornou-se também peça fundamental na mais nova plataforma móvel da Microsoft, o Windows Phone. Para tirar proveito de toda esta nova perspectiva, a comunidade Delphi tem à sua disposição o Embarcadero Prism XE2, que se utiliza dos recursos do .NET Framework ao mesmo tempo em que trabalha com uma linguagem de programação também baseada em Object Pascal, a Oxygene.

Evolução do Silverlight

Quanto à evolução do Framework em termos estruturais, as mudanças mais perceptíveis e significativas se deram ainda na transição de sua primeira para a segunda versão. A partir de então, um mesmo modelo foi seguido nas versões subsequentes até a mais recente. Apenas para não deixar de mencionar, as principais mudanças do Silverlight 1 para o 2 foram os acréscimos da CLR (Common Language Runtime), que é um subconjunto de classes do .NET Framework, o suporte a linguagens .NET (tais como C# e Oxygene) e um modelo de interface de usuário baseado no WPF. O Windows Presentation Foundation que é um subsistema do .NET Framework voltado para a construção de aplicações ricas Windows, provê então toda a base de um

XE2
mais recente versão do Silverlight

Fabrício Hissao Kawata

subconjunto de classes e controles ao Silverlight. A proximidade entre estas duas tecnologias ainda se dá pelo uso da mesma linguagem de marcação utilizada na definição das interfaces de usuário, denominada XAML (Extensible Application Markup Language), além de um mesmo sistema de animação e reprodução de mídias (áudio e vídeo).

Nota

Embora o Silverlight não seja suportado nativamente no Linux, há um projeto paralelo encabeçado pela equipe Mono (e com apoio da Microsoft), de uma implementação Open-source do Silverlight já conhecida do grande público, denominada Moonlight. Este que, popularmente, pode ser considerado o Silverlight para Linux, conta atualmente com uma versão preview que suporta o Silverlight 4, sem previsão para a versão 5.

Nota do DevMan

Line-of-Business Applications, ou simplesmente LOB Applications, são aplicações de missão crítica, providas de uma série de recursos vitais a um empreendimento, que engloba áreas fundamentais do negócio, tais como contabilidade, gestão, planejamento e de recursos. Uma característica comum a esse tipo de aplicação é a sua integração com banco de dados e sua conectividade via rede, seja interna ou pela Web.

Como mencionado, as mudanças significativas que ocorreram no Silverlight se deram ainda no seu início, no entanto, melhorias sutis e seletas não deixam de ocorrer a cada nova versão, bem como a correção de eventuais bugs, mantendo assim o Framework atrativo aos seus seguidores. Prova disso é que o Silverlight 5, traz algumas novas características, tais como:

- Suporte a navegadores 64bits;
- Desempenho melhorado, o que resulta numa maior rapidez em sua inicialização;
- Processamento de texto mais nítido para tamanhos menores;
- Acréscimo de uma nova porção portada do XNA (o Framework da Microsoft para a construção de jogos) que resulta, entre outras coisas, numa renderização de gráficos 3D acelerada por hardware, onde a placa gráfica se encarrega de todo o trabalho;
- Trick play, que permite reproduzir vídeos numa velocidade desejada (mais lento ou mais rápido), sem que a afinação do áudio seja alterada. Um exemplo clássico do uso deste recurso é quando se deseja assistir o vídeo de uma palestra longa, numa velocidade acelerada, para assimilar somente seus pontos mais interessantes;
- Pivot Viewer, um controle destinado à apresentação de coleções de dados, que combina imagens, animações e filtragem de dados para gerar uma experiência única de visualização;
- Aplicações Out-of-Browser agora suportam janelas secundárias (Child windows), da mesma forma que em aplicações Desktop tradicionais;
- XAML Debugging, recurso que permite depurar código XAML, visando principalmente o tratamento de erros relacionados à Data Binding (ligação de dados);

Para fixar conceitos, estimular o leitor iniciante e situar melhor o desenvolvedor mais experiente, a seguir são mostrados a fundo algumas destas e outras novidades do Silverlight 5. Isso faz com que o presente artigo seja uma fonte de informação que não se prenda tão somente no lado teórico ou no lado prático, mas sim uma unificação destes contextos.

Requisitos necessários para se trabalhar com o Silverlight 5

Já almejando a abordagem prática do uso do Silverlight 5 (SL5), o primeiro requisito necessário, até mesmo por questões óbvias, é possuir o ambiente de desenvolvimento a ser utilizado na construção das aplicações. O IDE em questão é o Prism XE2 (Visual Studio 2010), a mais recente versão do produto voltado para desenvolvimento .NET da Embarcadero. No entanto, o Prism, por si só, não vem preparado para a construção de projetos Silverlight. Para que isso seja possível, é necessária a instalação de um conjunto de ferramentas de Design e desenvolvimento, contidas em um único instalador denominado Siverlight 5 Tools (lembrando que o pacote é gratuito e seu link está disponível ao final do artigo). Este, todavia, requer uma instalação do Visual Studio 2010 já com o Service Pack 1 aplicado. Dessa forma, ao instalar diretamente o Prism XE2 no ambiente, apenas o Shell, que é uma versão compacta do Visual Studio, é instalado, o que não é o suficiente. Portanto, o ideal é que o ambiente possua um Visual Studio 2010 devidamente instalado já com o SP1 aplicado e, posteriormente, o Prism XE2 seja então adicionado. Ao fazer isso, o produto da Embarcadero se integra ao próprio IDE existente, na forma de plug-in.

No pacote do SL5 Tools já estão inclusos itens como o Runtime de desenvolvimento do Silverlight, chamado de Developer Runtime, responsável por executar as aplicações na máquina, bem como o seu kit de desenvolvimento padrão, ou seja, seu SDK (Software Development Kit). Vale lembrar que, caso seja de interesse, é possível instalar o pacote em questão relativo a uma versão anterior do Silverlight. Por fim, para verificar se o ambiente já está

realmente apto ao desenvolvimento com o Framework, basta tentar criar um novo projeto Silverlight. Neste momento, uma janela semelhante a da **Figura 1** deve ser exibida, com o detalhe da versão a ser trabalhada.

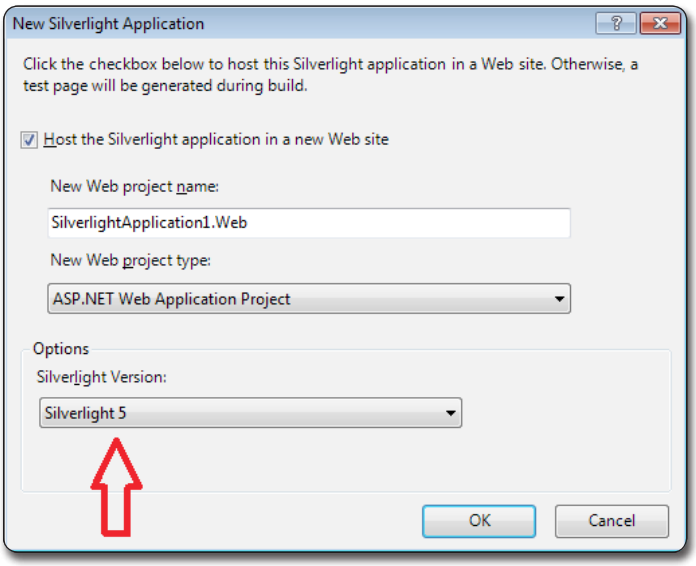


Figura 1. Silverlight 5 habilitado para desenvolvimento no Prism XE2

Vector printing (impressão de vetor)

O suporte a impressão no Silverlight é bem simplificada, e até considerada por muitos como sendo limitada. Isto porque, grande parte do trabalho envolvido é manual. Num caso trivial, supondo que a aplicação possua um conteúdo extenso que deva ser impresso, cabe ao desenvolvedor verificar se o mesmo se adapta à página a ser impressa e, caso não, ele é quem deverá também fazer a "quebra" deste conteúdo em quantas páginas forem necessárias. Inclusive as coordenadas da posição de cada controle também devem ser ajustadas. Para tentar simplificar um pouco as coisas, o Silverlight então faz uso do mesmo conceito de elementos, utilizado na criação da interface de usuário (UI) de suas aplicações, na impressão. Isso quer dizer que poderá é possível imprimir estes mesmos elementos que, ao invés de estarem em uma Page (página de um projeto de Silverligt), estarão renderizados num documento virtual de impressão.

Atualmente o Silverlight 5 suporta dois modos de impressão, a mais tradicional "impressão Bitmap" (Bitmap Printing), introduzida no Silverlight 4, e a mais recente "impressão de Vetor" (Vector Printing), que chega como uma das grandes novidades desta versão. Conceitualmente falando, a diferença básica entre as duas é que, enquanto a primeira envia uma imagem totalmente renderizada à impressora, e que por se tratar de um Bitmap possui suas limitações (tal como a grande quantidade de bytes), a segunda trabalha com uma impressão de mais alto nível. Isto porque, no modo Bitmap o Silverlight faz uso da classe WriteableBitmap para transformar cada página em uma imagem, suportando assim qualquer tipo de conteúdo. Pelo fato de tudo virar uma simples imagem, o modo Bitmap funciona em qualquer impressora. O grande "porém" neste caso fica por conta da velocidade, uma vez que uma grande quantidade de dados precisa ser transmitida à impressora. Já no modo Vetor, este desempenho é melhorado, já que o Silverlight passa para a impressora o processamento das texturas, formas e textos envolvidos.

Novamente visando facilitar a vida do desenvolvedor, o Framework faz uso de um único comando de impressão para ambos os tipos, permitindo que a escolha entre um ou outro se dê através de configurações da própria impressão (print settings). Para ver como isso tudo funciona, e sem se esquecer daqueles que estão iniciando em Silverlight e que provavelmente não conhecem sua parte de impressão, é possível ver o lado prático do contexto. Tradicionalmente, uma impressão comum no Silverlight se dá pelo envolvimento de uma instância da classe PrintDocument, do namespace System.Windows.Printing, e pela atribuição de um manipulador para o evento denominado PrintPage, responsável por especificar o conteúdo da página impressa. Fazendo menção então a uma página de uma aplicação hipotética simples, tome como exemplo a **Figura 2**, que demonstra uma Page contendo um elemento de imagem a ser impresso. Para ajudar no entendimento da explicação, o trecho principal do código XAML da estrutura é apresentado na **Listagem 1**.

Figura 2. Página de uma aplicação Silverlight com conteúdo a ser impresso

Listagem 1. Impressão - Código XAML da página de exemplo

```
01 <StackPanel x:Name="LayoutRoot">
02     <Image x:Name="ImgPrint" Source="Penguins.jpg"
03         Margin="10" Height="284" Width="412"
04         VerticalAlignment="Center" HorizontalAlignment="Center" />
05     <Button x:Name="BtnPrint" Content="Imprimir"
06         Margin="10" Height="25" Width="100"
07         Click="BtnPrint_Click" />
08 </StackPanel>
```

Em aplicações Desktop, principalmente no caso do Delphi, é muito comum que o recurso de impressão contido em um Form esteja centralizado em um botão, onde fica localizada a chamada a um determinado método Print de algum eventual componente de relatório. Isto é justamente o que acontece no exemplo apresentado,

cujo botão (BtnPrint) já possui um evento Click declarado (linha 7). No entanto, no seu code-behind, o método Print em questão não será relativo a um componente, e sim à já mencionada classe PrintDocument. A declaração do evento é mostrada a seguir, onde pdImpressão é uma variável que recebe uma instância de PrintDocument, e seu método Print recebe como argumento uma string que identifica o nome do documento de impressão (aquele que irá aparecer na fila de impressão).

```
method MainPage.BtnPrint_Click(sender: System.Object; e: System.Windows.
RoutedEventArgs);
begin
  pdImpressao.Print('ImgPrintTest');
end;
```

Comumente a instância em questão é criada no construtor da própria Page, para que fique disponível durante todo o ciclo de vida da mesma. Neste mesmo momento, é feita também a atribuição de um manipulador para o seu evento PrintPage, conforme exposto no trecho a seguir.

```
constructor MainPage;
begin
  InitializeComponent();
  pdImpressao := new PrintDocument();
  pdImpressao.PrintPage += PdImpressao_PrintPage;
end;
```

Por fim, para especificar o conteúdo a ser impresso, basta informar o elemento visual desejado para a propriedade PageVisual do objeto PrintPageEventArgs do método manipulador. O proposto exemplo imprime o elemento Image (ImgPrint) da página, todavia, poderia ser informado o elemento raiz StackPanel (LayoutRoot), fazendo com que a página por completo fosse impressa (imagem + botão).

```
method MainPage.PdImpressao_PrintPage(sender: System.Object; e: PrintPageEven-
tArgs);
begin
  e.PageVisual := ImgPrint;
end;
```

Ainda sobre o objeto PrintPageEventArgs, como foi dito no início, no Silverlight a "quebra" de um conteúdo muito extenso a ser impresso, tal como uma listagem de dados ou figuras, é um processo manual que cabe ao desenvolvedor. Justamente por isso, este objeto exerce um papel muito importante nesta parte do desenvolvimento, pois provê outras três propriedades úteis: PageMargins, PrintableArea e HasMorePages. A primeira retorna um objeto com detalhes das margens (Top, Bottom, Right e Left) da página. PrintableArea retorna um objeto com detalhes de altura e largura (Height e Width) da área de impressão disponível e HasMorePages tem valor booleano que indica se a impressão deve continuar em uma nova página ou não.

Por padrão, ao fazer a impressão efetiva do conteúdo, o próprio Silverlight é quem define se será utilizada a impressão Bitmap ou de Vetor, o que depende invariavelmente do driver da impressora e do tipo de conteúdo envolvido. Isto é feito por um mecanismo interno, denominado Fallback System, que prioriza a impressão de Vetor, desde que o dispositivo suporte PostScript e o conteúdo seja condizente. Este conteúdo se refere principalmente a certos efeitos utilizados que não são suportados, como o de transparência. A informação sobre qual modo a aplicação Silverlight irá trabalhar não fica disponível, e essa distinção só é percebida ao imprimir. Caso necessário, é possível ainda "forçar" o uso de determinado modo de impressão. No caso do Bitmap, basta utilizar o método PrintBitmap da classe PrintDocument, ao invés do Print. Já para o modo Vetor, deve-se utilizar uma instância da classe PrinterFallbackSettings que, dentre suas propriedades, apresenta uma de valor booleano denominada ForceVector. Esta mesma instância deve então ser passada como parâmetro ao método Print. Estes dois casos são mostrados na **Listagem 2**, com base no exemplo apresentado.

Listagem 2. Impressão – Forçando o modo Bitmap e Vetor

```
//Forçando - Bitmap
pdImpressao.PrintBitmap('ImgPrintTest');

//Forçando - Vector
var fb: PrinterFallbackSettings := new PrinterFallbackSettings();
fb.ForceVector := true;
pdImpressao.Print('ImgPrintTest', fb);
```

Nota do DevMan

PostScript é uma linguagem de descrição de páginas, originalmente criada para impressão, que define um padrão de encapsulamento de imagens . Foi criada pela Adobe e sua incidência é mais acentuada em dispositivos Mac.

ClickCount – Verificando quantidade de cliques do mouse

Uma novidade sutil do Silvelight 5, porém muito útil no cotidiano da interatividade proporcionada pelas aplicações, é o recurso voltado ao evento Click, relacionado ao mouse. Agora uma nova propriedade, intuitivamente denominada ClickCount, está disponível para que seja possível detectar um duplo ou triplo-clique. Até a versão 4, isto só era possível através de uma contagem manual, resultando nos famosos workarounds.

Olhando para o contexto das aplicações Silverlight, é garantido que a grande maioria, senão em sua totalidade, faz uso do clique do mouse, principalmente no que diz respeito ao estabelecimento da interatividade homem-máquina. Assim sendo, um evento muito utilizado é o MouseLeftButtonDown, disparado quando um elemento é clicado, através do clique de esquerda do periférico. Contudo, até então o Silverlight tratava este tipo de ação com base na convenção das interfaces de usuário voltadas para Web, que define a captura apenas do clique por si só. Agora no Silverlight 5, foi implantado de forma nativa o recurso de contagem de cliques. Na prática, suponha que uma página de determinada aplicação contenha um simples elemento Image que, ao receber um eventual duplo-clique, execute uma determinada ação. Do lado do código XAML haveria algo como a **Listagem 3**, onde o elemento já possui seu evento MouseLeftButtonDown declarado.

Já no code-behind, especificamente no código do manipulador do evento em questão, é que seria feita a verificação da quantidade de cliques, usando para isso a nova propriedade ClickCount do objeto MouseButtonEventArgs do método. Isto pode ser visto na **Listagem 4**. Só para não deixar de mencionar, ClickCount pertence ao namespace System.Windows.Input e declarado como somente leitura (por questões óbvias).

Listagem 3. Click Count – Código XAML de exemplo

```
<StackPanel x:Name="LayoutRoot">
    <Image x:Name="ImgClick" Source="Tulips.jpg"
        Margin="10" Height="284" Width="412"
        VerticalAlignment="Center" HorizontalAlignment="Center"
        MouseLeftButtonDown="ImgClick_MouseLeftButtonDown" />
</StackPanel>
```

Listagem 4. Click Count – Código Oxygene do manipulador do método MouseLeftButtonDown

```
method MainPage.ImgClick_MouseLeftButtonDown(sender: System.Object;
e: System.Windows.Input.MouseButtonEventArgs);
begin
  if (e.ClickCount = 2) then
    MessageBox.Show('Duplo-clique realizado!');
end;
```

Nesta mesma linha, ainda fazendo uso deste novo recurso, o desenvolvedor pode elaborar situações onde para cada determinada quantidade de cliques o aplicativo realize uma ação diferente. Um caso real clássico deste uso acontece no famoso processador de textos Microsoft Word. Por exemplo, num documento contendo texto, com um clique simples posiciona-se o cursor próximo à palavra desejada, com um duplo-clique a palavra é então selecionada e, finalmente, com um triplo-clique seleciona-se o parágrafo por inteiro. Portanto, elucidando esta situação e com base no mesmo exemplo anteriormente mostrado, tal verificação poderia ser feita conforme o trecho de código da **Listagem 5**. Neste caso, ao invés de uma mensagem em tela, usa-se um elemento TextBlock (TxbCount) para exibir a contagem, o que resulta em algo parecido com a **Figura 3**.

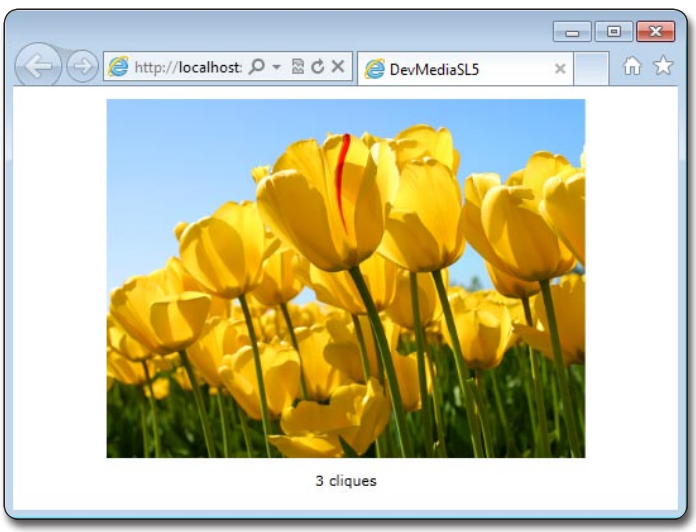


Figura 3. Triplo-clique detectado

Listagem 5. Click Count – Contagem de cliques

```
method MainPage.ImgClick_MouseLeftButtonDown(sender: System.Object; e:
System.Windows.Input.MouseButtonEventArgs);
begin
  if (e.ClickCount = 1) then
    TxbCount.Text := '1 clique'
  else if (e.ClickCount = 2) then
    TxbCount.Text := '2 cliques'
  else if (e.ClickCount = 3) then
    TxbCount.Text := '3 cliques';
end;
```

Vale lembrar que a velocidade do clique para que haja a distinção entre um clique simples, duplo ou triplo, depende do ajuste desta configuração no sistema operacional. Outra ressalva é quanto ao próprio uso deste recurso, pois dependendo da ocasião, podem ocorrer sobreposições de ações. Isto porque um clique acaba inevitavelmente fazendo parte de outro. Por exemplo, um clique simples, pode representar o primeiro de um triplo-clique. Dessa forma, casa haja ações distintas definidas para ClickCount = 1 e ClickCount = 3, um triplo-clique poderia fazer com que ambas fossem disparadas.

RichTextBlock e RichTextBlockOverflow

Na área de construção de interfaces de usuário, o Silverlight 5 apresenta dois novos controles baseados no existente RichTextBox, nomeados como RichTextBlock e RichTextBlockOverflow. Basicamente, ambos provêm uma nova capacidade de se trabalhar com a disposição da exibição de textos, possibilitando a criação de layouts mais rebuscados.

O RichTextBlock é um controle somente leitura, considerado um refinamento do tradicional RichTextBox. Trabalha em conjunto com um ou mais controles RichTextBlockOverflow, que ficam ligados a ele. Com isso é possível, por exemplo, "quebrar" um documento muito extenso em N pedaços menores, organizando-os de forma livre na página da aplicação. Um caso típico é a exibição de um texto divido em várias colunas, assim como vemos em publicações do mundo real, tais quais jornais e revistas. Resumidamente, na prática, quando o conteúdo a ser exibido pelo RichTextBlock é maior que sua área de exibição, haverá sobras que serão cortadas. Utilizando um RichTextBlockOverflow, pode-se então pegar este conteúdo de "sobra" e dispô-lo em outro local da página.

Exemplificando seu uso prático, suponha então uma situação onde seja necessário exibir de forma simples um texto de tamanho razoável em um controle RichTextBlock de certo tamanho, devidamente posicionado em uma página. Para facilitar a visualização deste novo recurso, o elemento raiz da estrutura é um Grid contendo uma única linha e três colunas. Ilustrando isso ao leitor, o trecho principal do código XAML da UI desta página é mostrado na **Listagem 6**. Vale lembra que o código completo está disponível junto ao código-fonte do artigo.

Listagem 6. RichTextBlock - Código XAML da página de exemplo

```
<Grid x:Name="LayoutRoot">
    <Grid.ColumnDefinitions>
      <ColumnDefinition></ColumnDefinition>
      <ColumnDefinition></ColumnDefinition>
      <ColumnDefinition></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition></RowDefinition>
    </Grid.RowDefinitions>
    <RichTextBlock x:Name="RtbText"
            Height="400" Width="200"
            TextAlignment="Justify"
            Grid.Column="0">
      <RichTextBlock.Blocks>
        <Paragraph FontSize="15" FontWeight="Bold" FontFamily="Verdana">
          <Run>TEXTO DE EXEMPLO</Run>
        </Paragraph>
        <Paragraph FontSize="10" FontWeight="Normal" FontFamily="Verdana">
          <Run>PARÁGRAFO 1 - Lorem ipsum...</Run>
        </Paragraph>
        <Paragraph FontSize="10" FontWeight="Normal" FontFamily="Verdana">
          <Run>PARÁGRAFO 2 - Nam a vulputate...</Run>
        </Paragraph>
        <Paragraph FontSize="10" FontWeight="Normal" FontFamily="Verdana">
          <Run>PARÁGRAFO 3 - Praesent eu tortor...</Run>
        </Paragraph>
      </RichTextBlock.Blocks>
    </RichTextBlock>
</Grid>
```

Supondo então que o conteúdo a ser mostrado, um texto com três parágrafos, não se encaixe devidamente na área do RichTextBlock envolvido, faz com que boa parte do texto seja "cortado" pelo controle, causando o efeito mostrado na **Figura 4**. Olhando para o código XAML apresentado e a imagem, fica fácil notar que uma parte do parágrafo 2 e o parágrafo 3 todo ficaram omitidos.

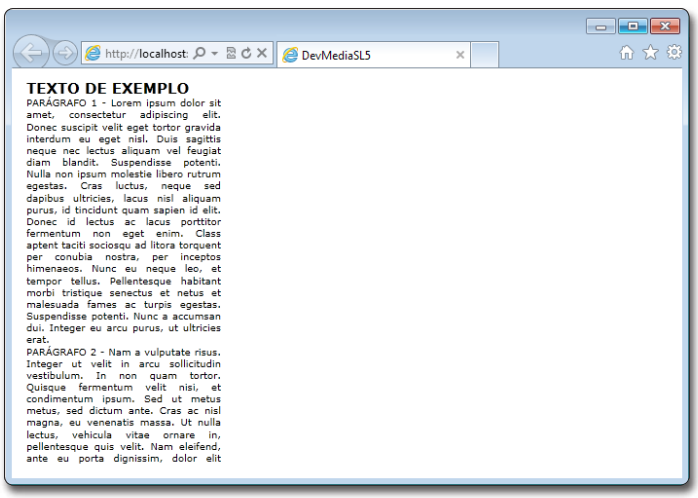


Figura 4. RichTextBlock - Conteúdo "cortado"

A solução neste caso vem através do novo recurso que, como já mencionado, conta com a combinação do já apresentado RichTextBlock, com o RichTextBlockOverflow. Dessa forma, o conteúdo "de sobra" poderia ser alocado em elementos desse tipo e então devidamente mostrado em tela na posição que desejar e que, neste caso, será imediatamente ao lado do texto já apresentado, ou seja, na segunda coluna do Grid. Com esta definição, a primeira medida é fazer a declaração do elemento RichTextBlockOverflow no código XAML que, para este exemplo, ficaria da seguinte forma:

```
<RichTextBlockOverflow x:Name="RtbOverflow"
                       Height="400" Width="200"
                       Grid.Column="1"></RichTextBlockOverflow>
```

No entanto, este elemento por si só não tem representatividade nenhuma no contexto da aplicação. Para isso, é necessário que ele esteja "linkado" a um RichTextBlock. Esta definição, ainda de acordo com o exemplo apresentado, faria com que a declaração do elemento RichTextBlock ficasse da seguinte forma:

```
<RichTextBlock x:Name="RtbText"
        Height="400" Width="200"
        TextAlignment="Justify"
        OverflowContentTarget="{Binding ElementName=RtbOverflow}"
        Grid.Column="0">
```

O detalhe aqui fica por conta da propriedade OverflowContentTarget, que recebe através do recurso de Binding o elemento RichTextBlockOverflow, estabelecendo o link necessário entre ambos. O resultado disso pode ser visto na **Figura 5**.

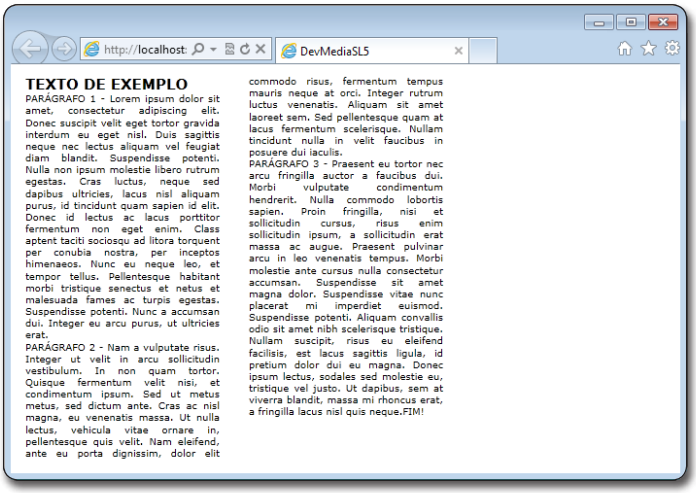


Figura 5. RichTextBlock combinado com RichTextBlockOverflow

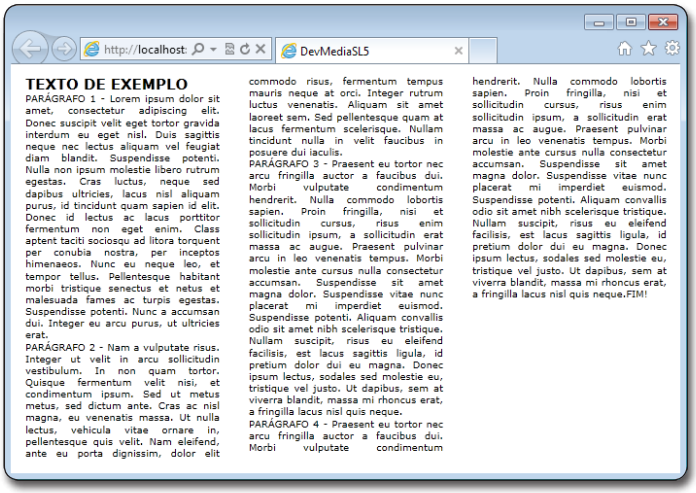


Figura 6. RichTextBlock combinado com dois elementos RichTextBlockOverflow

Conforme mencionado na introdução, o controle RichTextBlock pode estar vinculado a vários elementos RichTextBlockOverflow. Assim sendo, caso houvesse mais conteúdo "de sobra" a ser mostrado, a ligação seguinte não seria feita de forma direta como anteriormente, mas sim com a definição no próprio elemento RichTextBlockOverflow já vinculado, e assim sucessivamente. Isto resultaria num código XAML conforme a seguir, onde é mostrada a declaração de um novo elemento RichTextBlockOverflow (RtbOverflow2) e o vínculo deste com o RtbOverflow anteriormente existente, através da propriedade OverflowContentTarget.

```
<RichTextBlockOverflow x:Name="RtbOverflow"
            Height="400"Width="200"
            Grid.Column="1"
            OverflowContentTarget="{Binding ElementName=RtbOverflow2}">
</RichTextBlockOverflow>
<RichTextBlockOverflow x:Name="RtbOverflow2"
            Height="400"Width="200"
            Grid.Column="2">
</RichTextBlockOverflow>
```

Por fim, o novo resultado obtido é visto na **Figura 6**.

Melhorias no suporte a áudio e vídeo

A mais recente versão do Silverlight trouxe ainda algumas melhorias em seu já elogiado suporte a áudio e vídeo. Talvez a mais básica delas, porém não menos importante, é que a partir de agora, o próprio Silverlight impede que um vídeo que esteja em execução possa ser, em qualquer hipótese, interrompido pela proteção de tela (screensaver) do sistema. Isso previne o acontecimento de possíveis inconvenientes que prejudicavam a interatividade com o usuário final e que estava sujeita a acontecer até a versão anterior. Além disso, agora o Silverlight provê suporte a controle remoto, que facilmente são encontrados em computadores pessoais em razão de proporcionar uma maior comodidade ao usuário, podendo manipular à distância o áudio e o vídeo que estão sendo tocados no hardware. Todavia, o recurso que mais chama a atenção neste segmento é o que é chamado de Trick Play. Este "truque de reprodução" permite que se acelere ou desacelere um vídeo que está sendo executado, sem que o tom de seu áudio seja alterado. Com isso, o usuário é capaz, por exemplo, de assistir ao vídeo de uma palestra longa numa velocidade mais rápida que o normal, sem que o áudio deixe de ser devidamente compreendido pelo ouvido humano, ou seja, que não fique distorcido. Sem deixar de mencionar, apesar de menos usual, o Trick Play também pode ser utilizado também com arquivos de áudio.

Olhando para o recurso na prática, em termos de código tem-se pouco trabalho a fazer. Isso se resume à definição de um valor para a propriedade PlaybackRate do elemento MediaElement. Tal propriedade apresenta como valor padrão o número 1, que representa uma velocidade de execução normal da mídia. Assim sendo, fica fácil presumir que o número 2 fará com que a velocidade de execução seja dobrada e um valor menor que 1 fará com que a mídia seja executada numa velocidade proporcionalmente menor. Portanto, o código XAML de um elemento MediaElement que irá executar um arquivo de vídeo em velocidade "acelerada" seria algo como:

```
<MediaElement x:Name="Player"
        Height="300"Width="500"
        Source="Wildlife.wmv"
        PlaybackRate="2" AutoPlay="True">
</MediaElement>
```

Style Binding Expressions

Nesta sua versão de número 5, o Silverlight apresenta um novo recurso denominado Style Binding Expressions que, em suma, permite que suas técnicas de definição de Resources e Binding Expressions possam ser aplicadas e utilizadas com objetos Style. Dessa forma, através de Resources, os objetos de estilo (Style) podem se aproveitar de recursos definidos em seu contexto e a serem usados pelos elementos da UI, sendo que para essa ligação, utilizam-se definições de Binding Expressions. Um exemplo disso pode ser visto a seguir, onde é exibido um trecho de código XAML:

```
<UserControl.Resources>
    <FontStyle x:Key="fontRes">Italic</FontStyle>

    <Style x:Key="StyleTeste" TargetType="Button">
      <Setter Property="FontStyle" Value="{StaticResource fontRes}"></Setter>
    </Style>
</UserControl.Resources>
```

Uma vez declarado o objeto Style, cujo Setter faz uso de Binding Expressions para estabelecer o vínculo com um Resource anteriormente definido, qualquer elemento Button desta página poderá fazer uso deste estilo, tal como é mostrado no código XAML a seguir. O resultado final, em termos visuais, é um botão com texto em itálico (**Figura 7**).

```
<Button Style="{StaticResource StyleTeste}"
    Height="23" Width="120"
    Margin="50"
    Content="Botão teste">
</Button>
```

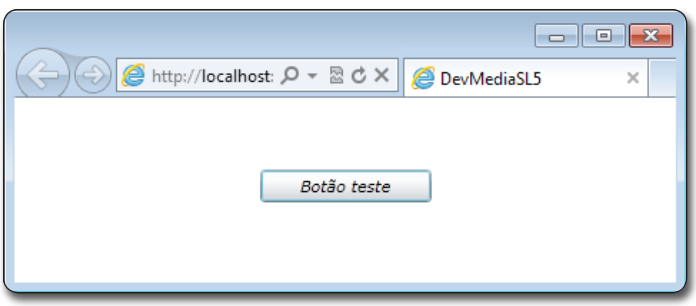


Figura 7. Uso de Style Binding Expressions

Conclusão

Como mostrado, o Silverlight 5 é a mais recente versão deste Framework da Microsoft que esta aí disponível para os desenvolvedores que queiram criar aplicações ricas de alta performance gráfica, acessível pela Web e com recursos adicionais incontáveis. Prova disso é a sua excelência em trabalhos com animações, multimídia (áudio e vídeo), gráficos e aceleração 3D. Para o desenvolvedor Delphi cabe apenas o bom senso para desfrutar de todas estas vertentes, através do Prism XE2, agregando novos nichos de negócios ao seu cotidiano profissional. Outra empolgação para se investir no estudo e aprofundamento da tecnologia, é que o Silverlight é figura principal na plataforma móvel Windows Phone e que, apesar de sutis diferenças entre suas distribuições Web e Mobile, os conceitos básicos são os mesmos, havendo assim um reaproveitamento de conhecimento.

Se você acompanha regularmente esta revista, pode comprovar que tais assuntos como Silverlight, Prism e Windows Phone são bem recorrentes, o que lhe dá uma boa fonte de informação, seja para iniciar ou continuidade do estudo. A sugestão da vez então fica por conta das vídeo-aulas disponíveis no portal DevMedia, especificamente os cursos de Prism, que são uma boa oportunidade para se conhecer as nuances da linguagem Oxygene e detalhes do IDE Prism/Visual Studio.

Agradeço a todos e até a próxima.

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Blog: prismsilver.blogspot.com
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 6 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas.

Embarcadero Prism XE2 Enterprise - Trial
https://downloads.embarcadero.com/free/prism

Microsoft Silverlight 5 – Página oficial
http://www.silverlight.net

Silverlight 5 Tools
http://www.silverlight.net/downloads

Moonlight - Página do projeto
http://www.mono-project.com/Moonlight

Service Pack 1 do Microsoft Visual Studio 2010
http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=23691

Microsoft Visual Studio 2010 – Trial
http://www.microsoft.com/visualstudio/en-us/try

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Gerenciando banco de

Conheça o Rapid SQL XE2

O Rapid SQL pode ser definido como o IDE (Integrated Development Environment) da Embarcadero para a manutenção efetiva de um banco de dados, mais especificamente de seus objetos. Dessa forma, entenda que tarefas como criar, editar e versionar tais objetos, além da produção de códigos SQL relacionados, são feitas em uma interface de usuário muito intuitiva, que prima por facilitar a vida do profissional que estará em constante contato com a ferramenta.

Diferentes são os profissionais envolvidos com o desenvolvimento em si e que podem tirar proveito dos recursos providos pela ferramenta. Como, por exemplo, o DBA que consegue otimizar seu tempo no que diz respeito à realização de suas tarefas, que se tornam mais ágeis pelos diversos wizards (assistentes) e editores disponíveis para distintas funções. Outro ponto é a utilização de um ambiente único e centralizado para a administração de bancos de dados de plataformas diferentes. Já o desenvolvedor consegue, sem muito esforço e conhecimento, gerar o código SQL de que necessita, para plataformas diversas. Além disso, a construção de uma base de dados completa, seja para testes ou produção, se dá de forma totalmente visual e trivial. Por tudo que o ambiente proporciona, adicionado de sua integração com um sistema de controle de versão, a figura do Project Manager (gerente) fica apto a realizar de forma eficaz o gerenciamento de N projetos, no que diz respeito à parte de banco de dados.

Atualmente o Rapid SQL se encontra em sua versão XE2 (8.0.2, até o momento da escrita deste artigo) e surgiu como um dos produtos da família XE2. Tal lançamento teve como destaque a nova e badalada versão do Delphi, que acabou por ofuscar os demais produtos. Até mesmo por isso, olhando pelo lado do programador Delphi, esse ofuscamento culmina com o desconhecimento da existência da ferramenta e, consequentemente, uma não utilização por boa parcela da comunidade. Mas neste artigo vamos conhecer suas funcionalidades.

Resumo DevMan

De que trata o artigo:

O Embarcadero Rapid SQL XE2 é uma ferramenta voltada para profissionais desenvolvedores e administradores de banco de dados (DBAs). Através dela, é possível fazer a manipulação de bancos de dados das principais e mais populares plataformas existentes atualmente no mercado, tais como Firebird, Oracle, Interbase, MySQL e Microsoft SQL Server. Estes consequentemente fazem parte do cotidiano de qualquer ambiente de desenvolvimento. A ferramenta ainda auxilia, entre outras coisas, na produção de códigos SQL de alta performance que acabam por acelerar o ciclo de sucesso de aplicações.

Em que situação o tema é útil:

Os desenvolvedores em geral, sejam autônomos ou empregados, têm ou já tiveram a responsabilidade de manter uma base de dados de determinado projeto de software. Se na teoria aprendemos que funções como esta deveriam ser executadas por profissionais específicos, na prática acabamos por presenciar outra realidade. Sendo assim, é imprescindível que o desenvolvedor obtenha conhecimento de ferramentas que possam auxiliar neste tipo de trabalho, de forma a ganhar tempo e aumentar sua produção. Facilidade de uso, alta gama de recursos diversos e uma interface amigável são quesitos que se esperam encontrar nestas ferramentas, e o Rapid SQL XE2 consegue preencher todos eles de forma elegante, conforme poderá ser visto em sua apresentação no presente artigo.

Gerenciamento de banco de dados:

Recentemente, a Embarcadero apresentou ao mercado sua nova família de produtos, denominada XE2, que é segmentada nas diversas áreas do desenvolvimento de software. Só para citar, nestas incluem-se construção de aplicativos (Desktop, Web e Mobile), banco de dados (administração, performance e produção), modelagem e arquitetura. Dentre os produtos, o Rapid SQL é aquele destinado à manutenção de banco de dados e que se torna importante perante o cotidiano de qualquer contexto de desenvolvimento. Neste cenário pode-se inserir o Delphi, cuja comunidade é o foco principal desta revista. Assim sendo, citando uma aplicação tradicional VCL Forms, é comum seu envolvimento com banco de dados, seja qual for a plataforma deste (Interbase, Firebird, Oracle, MySQL, Microsoft SQL Server, entre outros), provido pelo suporte multibanco do framework nativo dbExpress. O Rapid SQL permite então que se gerencie este emaranhado de opções diferentes através de um IDE único e centralizado.

Características técnicas e instalação

O Rapid SQL XE2 é uma aplicação nativamente 32 bits e essencialmente Windows, isso quer dizer que ela irá rodar somente em sistemas operacionais Microsoft Windows (XP, Vista e 7), em distribuições tanto 32 quanto 64 bits. Sua lista de plataformas de banco de dados suportados é bastante generosa, conseguindo atender tanto as necessidades de um profissional de banco de dados, quanto à de um desenvolvedor (principalmente Delphi), conforme é mostrado a seguir:

- Interbase (2007 e 2009);
- Firebird (2.0);

Fabrício Hissao Kawata

- Microsoft SQL Server (2000, 2005 e 2008);
- MySQL (4.x);
- Oracle (8i, 9i, 10g e 11g);
- Sybase ASE (12.5 – 15.0.3);
- IBM DB2 for z/OS (8 e 9);
- IBM DB2 for Linux/Windows/Unix (8.0 – 9.7).

Em termos de instalação o Rapid SQL mantém o mesmo padrão dos outros produtos da Embarcadero, que são providos de um assistente (Installation Wizard) que guia o usuário passo-a-passo, até o término da operação. Um detalhe a mencionar é que mesmo o produto sendo comercial, pode ser baixado gratuitamente e funcionar sob uma licença de avaliação (Trial), limitada em 15 dias. Para isto é obrigatório que a pessoa tenha um cadastro ativo no site da fabricante, pois sua credencial será exigida no momento da instalação do software para o registro do mesmo.

IDE

Uma vez instalado, instantaneamente no momento de sua primeira inicialização, o Rapid SQL já procura facilitar a vida de seu utilizador oferecendo, através de uma caixa de diálogo nomeada como Quick Start Setup (**Figura 1**), a configuração dos Data Sources (fontes de dados). Em outras palavras, os bancos de dados que você irá manipular e manter através da ferramenta e que, por este motivo, deverão estar devidamente registrados nela. Este processo pode ser feito então de duas maneiras: manual e automática. A primeira é a forma mais tradicional, onde se deve informar o tipo de banco de dados, seu tipo de conexão, entre outros dados relacionados. Estes dados também podem ser obtidos através da importação de arquivos de configuração provenientes de outras ferramentas da própria Embarcadero. Já a automática é uma peculiaridade do Rapid SQL, em que envolve a utilização de um recurso denominado Discover Datasources, que efetua uma busca automática por arquivos de configuração de SGBDs presentes na máquina e, uma vez que encontra, faz o registro automático das bases de dados definidas nele.

Após esta etapa inicial, o IDE é então mostrado em tela. A **Figura 2** exibe uma imagem geral deste ambiente. Através dela podemos notar a presença da sua lista de menus e de sua barra de tarefas ao topo. Na lateral, sua janela organizadora de fontes de dados e mais abaixo suas guias adicionais de informações, tais como mensagens e depuração. Tudo isto vem a mostrar sua nítida proximidade com o IDE do próprio Delphi, tornando seu uso mais intuitivo para este profissional.

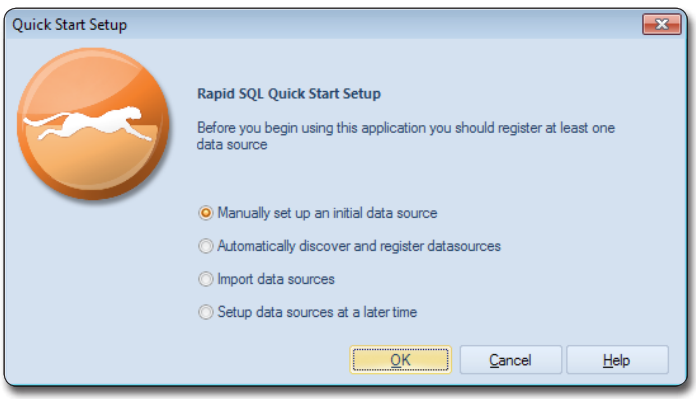


Figura 1. Quick Start Setup

Nota do DevMan

SGBD é a sigla para Sistema de Gerenciamento de Banco de Dados. Conceitualmente falando, pode ser definido como um agrupamento de programas de computador incumbidos de gerenciar uma base de dados. Isto significa controlar o acesso aos dados armazenados, bem como efetuar sua manipulação e organização. São exemplos de SGBDs: Firebird, MySQL, Oracle, SQL Server, PostgreSQL, Interbase, entre outros.

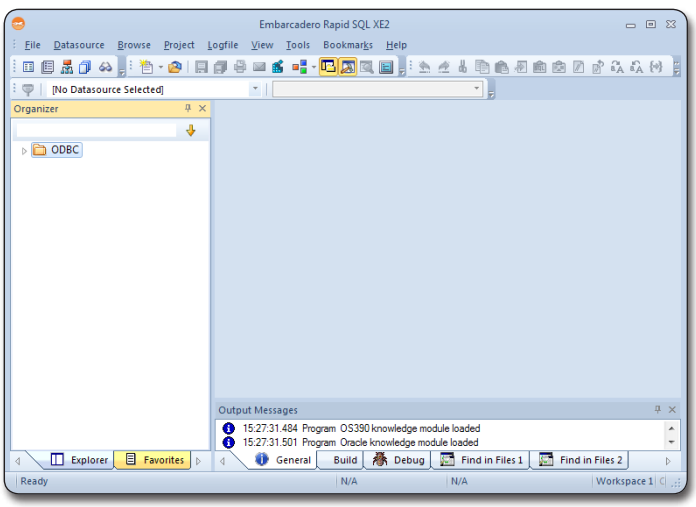


Figura 2. Overview do ambiente do Rapid SQL XE2

Database Explorer

Da mesma forma que o Delphi possui uma guia chamada Data Explorer, o Rapid SQL possui uma estrutura semelhante denominada Database Explorer presente na janela Organizer, responsável por dispor e organizar os objetos de um banco de dados.

A principal diferença aqui é o seu grau de importância perante o contexto. Enquanto no Delphi a estrutura funciona como um eficiente apoio ao trabalho de programação envolvido, no Rapid SQL ela é nada menos que o ponto de partida para grande parte de suas funcionalidades, sejam elas triviais ou avançadas.

Vendo sua parte organizacional, a **Figura 3** mostra uma base de dados Firebird devidamente registrada no ambiente e sua árvore de objetos sendo mostrada.

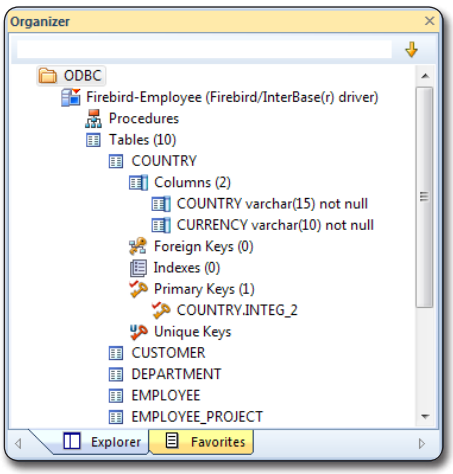


Figura 3. Database Explorer

A escolha pela citação do Firebird se deve ao fato deste SGBD ser um dos mais utilizados pela comunidade Delphi, principalmente em projetos tradicionais Win32. Além disso, essa predileção se deve ao fato de que o Firebird, além de gratuito, mostra-se muito robusto em termos de recursos e com uma manutenção facilitada até mesmo por profissionais não especialistas em banco de dados, como é o caso dos próprios programadores. No Rapid SQL XE2 o registro de um banco Firebird é realizado utilizando-se uma conexão nativa ou ODBC (que é o caso exemplificado na figura).

Criando novos objetos

Com base no que já foi mencionado anteriormente, a criação de novos objetos de banco de dados, sejam quais forem, pode ser realizada diretamente na árvore estrutural exibida na Database Explorer. Para realizar este tipo de operação o Rapid SQL provê ao usuário eficientes assistentes, os famosos Wizards, que irão guiá-lo a cada etapa da criação, descartando qualquer necessidade do usuário ter que digitar qualquer código SQL durante o processo. Elucidando um cenário real de manipulação de um banco de dados relacional, é imprescindível o uso de tabelas. Tais estruturas representam um conjunto de dados dispostos em linhas e colunas. Linhas são os registros de valores relacionados à cada coluna, em que está especificado o nome e o tipo de dado a ser recebido. As colunas também são conhecidas como "campos" de uma tabela.

Na prática, o simples acesso ao item do menu de contexto (através do botão direito do mouse), denominado Create, do nó Tables da estrutura do banco de dados, já possibilita a criação de uma nova tabela, ocorrendo a chamada imediata ao Wizard de apoio específico (**Figura 4**).

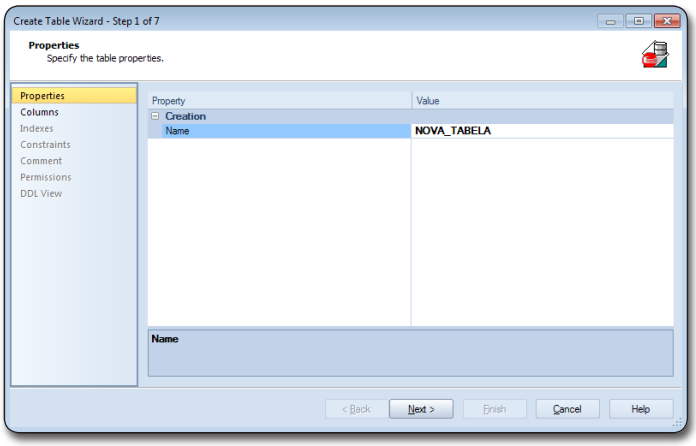


Figura 4. Wizard para a criação de tabelas

Conforme pode ser notado, o assistente em questão possui sete passos, cada qual define uma parte do objeto que está sendo montado, tais como colunas, índices, comentários, permissões e, por fim, todo o código SQL que o próprio Rapid SQL gera e que deverá ser executado para efetivar a criação do objeto. Da mesma maneira, a criação de outros objetos tais como Views, Stored Procedures e Triggers, seguem essa mesma linha, tendo como ponto de partida o menu de contexto do próprio nó referente ao objeto que se deseja.

Editando objetos existentes

Com a mesma intenção de facilitar ao máximo o trabalho do usuário, o IDE provê também uma forma bem intuitiva para fazer alterações em um objeto existente. O início deste processo se dá pelo menu suspenso do objeto que se deseja editar e posteriormente selecionando a opção Open. Ao fazer isso, uma nova janela é aberta, mais precisamente o editor de objetos (Object Editor) do Rapid SQL. As informações e parâmetros dos objetos são dispostos em diversas guias internas à janela, que ficam separadas por funcionalidade. Para cada tipo de objeto a janela do editor é customizada de uma forma diferente, principalmente no que diz respeito a estas guias. Adicionalmente, o Object Editor apresenta uma barra de ferramentas contendo algumas ações úteis ao gerenciamento do objeto, tais como criar, alterar, excluir ou extrair seu código SQL de definição. Toda esta estrutura citada pode ser vista na **Figura 5**, que mostra o editor para um objeto do tipo Generator. O tipo generator, ou sequence, é um objeto do banco de dados que é utilizado para manter uma sequência numérica. Vale lembrar que este mesmo tipo de janela é a que será utilizada para a edição dos objetos de qualquer banco de dados das plataformas suportadas pelo Rapid SQL. Isso faz com que o simples conhecimento do uso da ferramenta em si seja o suficiente para manipular bases Oracle, MySQL ou Interbase (só para citar algumas). Não é necessário conhecer a fundo as nuances de cada

SGBD, deixando este trabalho de fundo a cargo do IDE, ficando o usuário concentrado apenas no negócio.

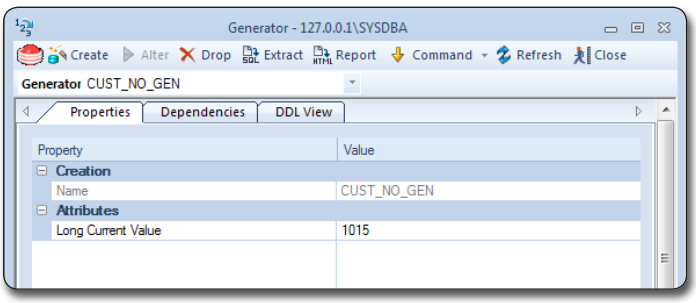


Figura 5. Object Editor

Outra grande habilidade do editor de objetos é o seu trabalho com alterações estendidas, ou seja, mudanças em um objeto que tenham influência sobre dados, permissões e a estrutura de outro(s) objeto(s). Em função disso, em alterações deste tipo, o Rapid SQL faz uma análise da estrutura atual do banco de dados, detectando os pontos que serão afetados e, por conseguinte, gerando automaticamente todo o código do script SQL necessário para que a alteração possa ser executada com sucesso. No entanto, como não poderia deixar de ser, a ferramenta preserva as regras de integridade existentes.

Visualizando dados contidos em objetos

Objetos de banco de dados tais como tabelas e visões (views) podem conter ou retornar dados armazenados dessa forma, é imprescindível a visualização destes na ferramenta de trabalho de seu cotidiano, tal como o Rapid SQL. Nele, este recurso está disponível sob o intuitivo nome de Select * From, presente no menu de contexto dos objetos citados. Assim que é acionado é aberta uma nova janela contendo duas abas (Query e Results), onde esta segunda exibe os dados dispostos na forma de Grid. A **Figura 6** mostra um exemplo do recurso exibindo dados da tabela Customer, do banco Employee do Interbase XE.

Na prática, sem deixar de mencionar a aba Query, é nela que se encontra o comando Select * From (que dá nome à opção) direcionado à tabela ou view relacionada.

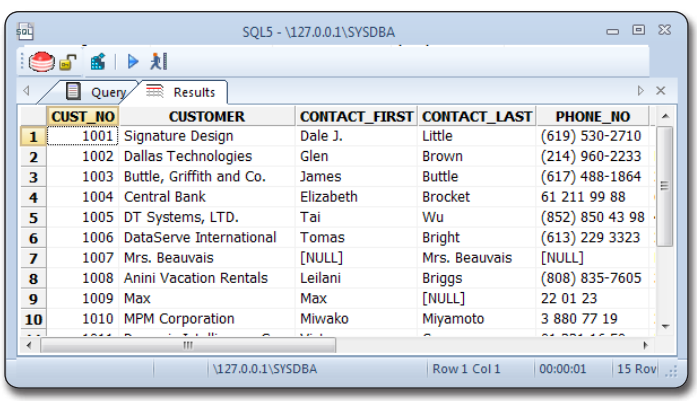


Figura 6. Visualizando dados de uma tabela

Editando dados

Para determinados tipos de objetos, novamente citando as tabelas (Tables), além de sua estrutura há a necessidade em se manter e manipular seus dados armazenados. Entenda por manipular as ações de inserir, alterar e excluir informações. No Rapid SQL este tipo de atividade é realizado através de um recurso denominado Data Editor, que quando acionado exibe um editor de dados que provê todas as ações necessárias citadas. Imediatamente no momento de sua execução, é aberta uma caixa de diálogo de filtros (Data Editor Filter), onde deverão ser determinadas informações que incidirão diretamente sobre os dados que estarão disponíveis para manipulação no Data Editor. Em outras palavras, neste momento deverá se informar as colunas (campos) da tabela a serem exibidos e possíveis condições de restrição (Where). O Data Editor trabalha em dois modos, Live e Batch. Em Live, as mudanças sobre os dados são efetivadas no mesmo momento de trabalho, assim que a ação de Salvar é chamada (seja pelo botão específico ou pela simples mudança de linha do registro). Já em Batch, as mudanças não são efetivadas no momento em que ocorrem. Ao invés disso, a cada modificação efetuada sobre os dados a instrução SQL responsável por efetuá-la é direcionada à guia SQL, de forma acumulativa. Dessa forma, num segundo momento, todas as instruções poderão ser então executadas em lote. Explicitando esta situação, a **Figura 7** mostra o momento de uma edição de dados no modo Batch, onde em seu rodapé podem ser visualizadas as instruções acumuladas a serem executadas.

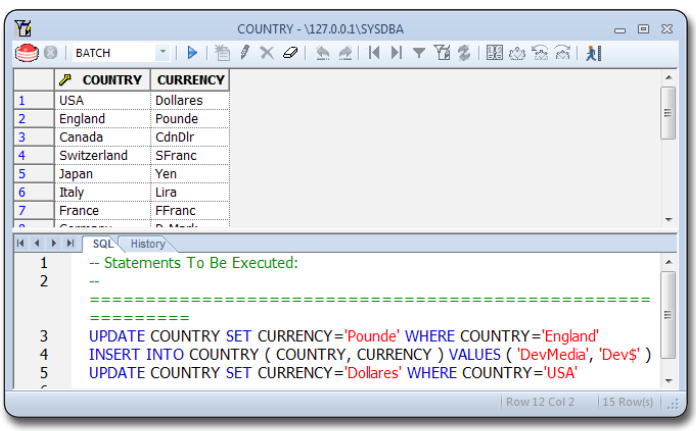


Figura 7. Data Editor em modo Batch

Documentando objetos

Na medida em que se faz a manutenção de um banco de dados, é fato surgirem situações onde seja necessária a documentação dos objetos contidos na base, seja por motivos gerenciais, de comunicação ou até mesmo pela manutenção de histórico. Para este tipo de documentação, principalmente em termos de comunicação é fundamental que o documento elaborado esteja num formato que permita sua leitura pelas mais diferentes pessoas, cada qual utilizando seu próprio ambiente de trabalho do cotidiano. Dessa forma, o ideal é que tal formato seja de caráter universal como, por exemplo, simples documentos HTML que possibilitam a

visualização de seu conteúdo em qualquer navegador de internet, seja qual for o sistema operacional. Em vista disso o Rapid SQL XE2 disponibiliza uma forma rápida e eficiente para a geração de relatórios contendo informações de qualquer objeto presente em um banco de dados registrado em seu IDE. Informações essas que incluem: tipo do objeto, estrutura, permissões, dependências, referências, código SQL de definição (DDL), etc. Este recurso é acionado ainda na guia Explorer (Database Explorer) da janela Organizer, através da opção Report do menu de contexto do objeto que se deseja. Uma vez acionada uma janela de diálogo é exibida, onde deverão ser informados o nome do arquivo que será gerado, bem como o título do documento. O resultado desta ação é visto na **Figura 8** que mostra um exemplo de relatório HTML gerado para a tabela Department, do banco Employee que acompanha a instalação do Firebird.

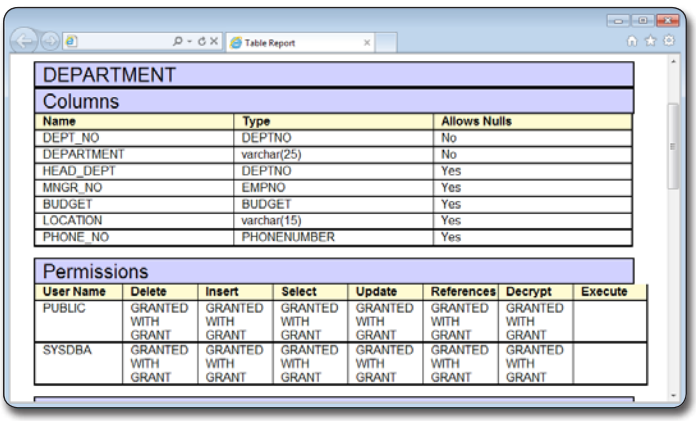


Figura 8. Relatório HTML de um objeto do banco de dados

Nota do DevMan

Em banco de dados, DDL (Data Definition Language) é uma linguagem de definição de dados, determinada como um subconjunto da linguagem SQL. Através dela define-se e manipula-se toda a estrutura de um banco de dados, tais como suas tabelas, linhas, colunas, índices e outros objetos através de comandos como Create (criar), Alter (alterar) e Drop (remover).

Favoritos

No IDE do Rapid SQL XE2, em sua janela Organizer, ao lado da guia Explorer (Database Explorer) existe outra denominada Favorites. Conforme seu nome sugere o usuário pode fazer neste local suas "marcações" favoritas e então armazená-las. Entenda como marcação um trecho de código SQL, VBScript ou JScript para qualquer fim, desde um simples "select" até alguma função elaborada. Uma vez armazenadas permanecem no ambiente, funcionando assim como uma biblioteca de recursos prontos a serem utilizados a qualquer momento. Isso vale para qualquer SGBD suportado, sendo que as marcações de cada um ficam devidamente agrupadas, mantendo uma organização. A **Figura 9** mostra a guia Favorites com os itens nativos para as plataformas Interbase e Firebird.

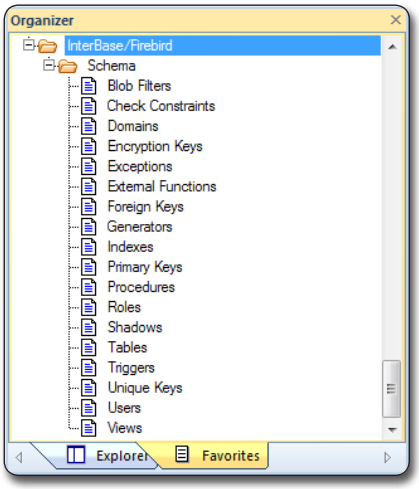


Figura 9. Favorites – Opções para Interbase/Firebird

Com um clique de direita sobre algum item da árvore, através dos comandos Open ou Execute, é que se dá a utilização de cada um destes itens favoritos. Para o primeiro caso a estrutura selecionada é então aberta numa nova janela (SQL Editor) da ferramenta, trazendo o código SQL armazenado já pronto para ser executado. Ideal para os casos onde se deseja visualizar o trecho a ser executado seja por precaução ou para realizar eventuais alterações. Já utilizando o Execute, a execução do código SQL é direta, sendo aberta posteriormente somente uma janela trazendo os resultados. Para ter uma ideia do tipo de script que pode ser armazenado, tome como base o item Tables da estrutura para Interbase e Firebird mostrada na figura anterior. Sua instrução SQL é mostrada na **Listagem 1**. O desenvolvedor que possua um conhecimento mediano nestes SGBDs pode facilmente notar que o código em questão irá retornar uma listagem contendo o nome e o tipo (Persistent ou Global) das tabelas presentes no banco de dados.

Listagem 1. Código SQL do item Tables dos favoritos para Interbase e Firebird

```
SELECT
  T.RDB$RELATION_NAME,
  T.RDB$RELATION_TYPE
FROM RDB$RELATIONS T
WHERE RDB$RELATION_TYPE IN ('PERSISTENT','GLOBAL TEMPORARY')
ORDER BY T.RDB$RELATION_NAME
```

Para adicionar um novo item aos favorito, é necessário possuir um arquivo de script contendo o código que se deseja armazenar. Este será indicado então através da opção New Item do nó de alguma das plataformas de banco de dados disponíveis.

File Execution Facility

File Execution Facility é um recurso do Rapid SQL que permite que arquivos de scripts SQL possam ser adicionados à sua estrutura para então poderem ser executados, tanto de forma imediata quanto de forma agendada, incidindo em uma ou várias bases

de dados ao mesmo tempo. Ele fica localizado no menu Tools do IDE e uma vez acessado, exibe uma janela conforme mostra a **Figura 10**.

Figura 10. File Execution Facility

Como pode ser visto seu layout é bastante intuitivo, no entanto, cabem algumas considerações. Em sua aba Files é onde deverão ser indicados os arquivos de scripts a serem executados. Em Target serão selecionados os Datasources de destino, ou seja, as bases de dados registradas no IDE em que a execução se dará. Uma vez executado, o código produzirá resultados que poderão ser exibidos tanto na janela de resultados da ferramenta quanto em arquivos externos (HTML, XML, etc.), sendo esta configuração feita na aba Output. Com o término do processo, principalmente nos casos de agendamento da tarefa e também da utilização de vários arquivos de script em várias bases diferentes, o ideal é que as pessoas interessados sejam notificadas. Isto é conseguido através da aba Notify, que permite a inserção de endereços de e-mail ou de rede dos envolvidos. O agendamento citado é configurado através da opção de Schedule, onde se determinam data e hora de execução.

Script Execution Facility

Da mesma forma que o Rapid SQL permite a execução de arquivos de scripts SQL em várias bases de dados simultâneas, ele provê também um recurso para o trabalho somente com os scripts em si. Em outras palavras, nesta situação não há a necessidade da presença de arquivos uma vez que o usuário irá digitar diretamente no Script Execution Facility o código SQL que deseja executar. Este recurso também está disponível no menu Tools e sua interface é muito próxima do File Execution Facility, conforme pode ser visualizado na **Figura 11**. Tirando o fato de não trabalhar com arquivos, os restantes das opções em tela são exatamente iguais às apresentadas anteriormente para File Execution Facility.

Facilitando a utilização do IDE

Com o passar do tempo, à medida que se adota o Rapid SQL como ferramenta principal para a gestão de bancos de dados em seu trabalho, é provável que em determinado momento acabe se tendo um emaranhado de Data Sources registrados na Database Explorer. Citando como exemplo uma empresa de softwares, nela poderia haver a existência de projetos atrelados a SGBDs distintos, tais como Firebird, MySQL e Oracle, variando de acordo com as preferências e disponibilidades dos clientes. Dessa forma, o profissional envolvido com toda essa gestão necessita de, além de recursos voltados à manipulação das bases de dados, outros que facilitem e agilizem o seu trabalho com o IDE propriamente dito. Nesta linha de pensamento o Rapid SQL provê alguns recursos simples, mas não menos importantes, tais como: Bookmarks, Recent Datasources e Retain View Settings.

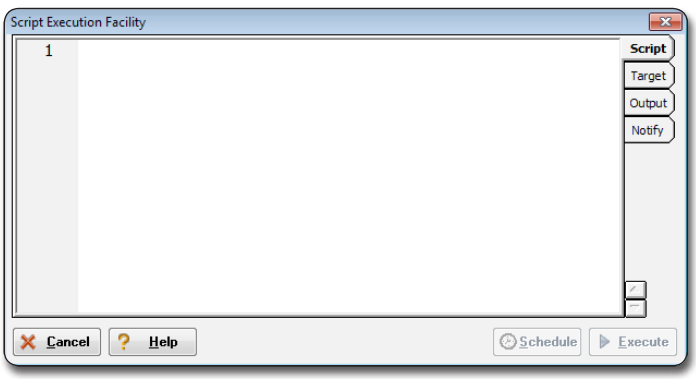


Figura 11. Script Execution Facility

O recurso de Bookmarks permite que o usuário "marque" aqueles objetos dos bancos de dados registrados no Rapid SQL que ele mais utiliza em seu dia-a-dia ou em determinado período. Dito isto, suponhamos que em certo semestre do ano os novos projetos de software em que se trabalha envolvem dois bancos de dados de plataformas diferentes, Interbase e Oracle, e mais precisamente três novas tabelas em cada um. Assim sendo, a cada dia de trabalho o usuário teria que expandir na Database Explorer as conexões de cada plataforma e então procurar os referidos objetos em cada base. Para evitar este tipo de trabalho "braçal" diário e utilizando o recurso de Bookmarks do Rapid SQL, basta que para cada objeto que se queira seja acionado sua opção Add Bookmark presente no menu de contexto. Cada bookmark pode então receber um nome fictício que facilite sua identificação futura. Uma vez feito isso, sempre que for necessário acessar este objeto basta acioná-lo pelo menu Bookmarks do IDE. Cada marcação salva se mantém agrupada por plataforma conforme mostra a **Figura 12**, que exibe duas marcações para o SGBD Firebird. Neste mesmo menu, é disponibilizado ainda um gerenciador de marcações (Bookmark Manager), que visa facilitar o controle de bookmarks na medida em que se tornam numerosos. Através do gerenciador podem ser aplicados filtros para a localização de determinada marcação ou ainda renomeá-las ou excluí-las.

Continuando, outro recurso citado é o de Recent Datasourc, que está disponível no menu File da ferramenta, e que como sua própria nomeação sugere faz referência aos Data Sources utilizados mais recentemente no ambiente do Rapid SQL. Assim, toda vez que iniciar o IDE o usuário pode simplesmente navegar por esta opção e prontamente acessar uma das bases de dados em que esteve trabalhando no dia anterior, por exemplo. No momento

em que este acesso rápido é acionado, escolhendo-se determinado Data Source, o Rapid SQL prontamente expande o seu nó na Database Explorer, bem como já ativa sua conexão, deixando-o pronto para uso.

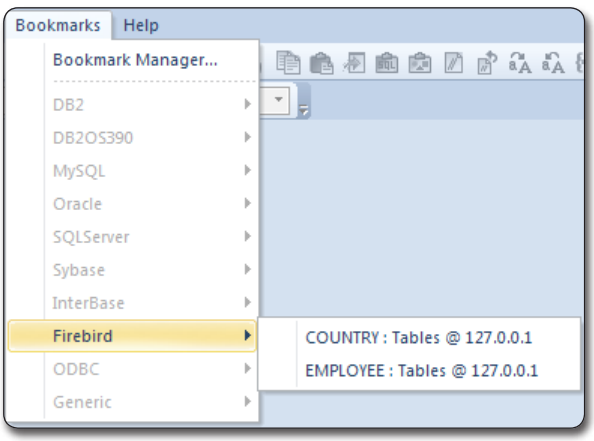


Figura 12. Bookmarks

Por fim, o último recurso mencionado é o denominado Retain View Settings, presente na guia Explorer da janela Organizer e seu acesso é feito através de um botão cujo ícone tem a forma de seta (**Figura 13**). Este recurso pode ficar marcado ou desmarcado. Quando marcado, faz com que a visualização da guia Explorer (Database Explorer) seja mantida e assim permaneça no momento em que o IDE seja aberto da próxima vez. É como se uma foto de sua estrutura naquele momento fosse tirada. Isso quer dizer que todos os nós que estavam expandidos, bem como todas as conexões que estavam ativas no momento em que se fechou a ferramenta se encontrarão desta mesma maneira ao se abrir o Rapid SQL novamente. Este simples recurso evita o seu trabalho exaustivo e repetitivo diário, de ativar e expandir a estrutura de cada base em que se estava trabalhando até o fim do expediente do dia anterior.

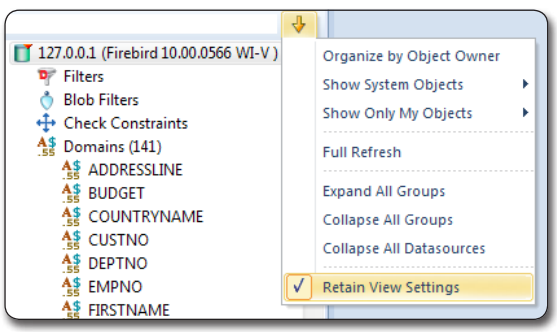


Figura 13. Retain View Settings

Gerando código automaticamente

O Rapid SQL XE2 pode produzir ainda, de forma automática, o código SQL necessário para a definição de instruções de Select, Insert, Update e Delete. Tais instruções podem vir na forma de simples Statements, como também através de construções de objetos Stored Procedures. Para isso este recurso faz uso de outros objetos de bancos de dados, mais especificamente tabelas (Tables) e visões (Views) pertencentes aos Data Sources registrados. De acordo com a escolha do usuário, todo o código necessário para determinada finalidade é produzido automaticamente. Assim sendo, em ambos os casos o código SQL resultante é imediatamente copiado para uma nova janela SQL Editor, cabendo ao usuário somente executar tal instrução. A opção por Statements resulta em simples instruções SQL enquanto que para Stored Procedures é gerado o código DDL de sua estrutura.

No IDE há basicamente duas formas para se realizar tal processo. A primeira é através da opção Generate, disponível no menu suspenso dos objetos mencionados (tables e views). Esta opção é subdividida em dois grupos, Stored Procedures e Statements, cada qual com a possibilidade de criação de código para Select, Insert, Update e Delete. Uma vez selecionada uma dessas opções, é aberta uma nova janela para a definição da estrutura do código que se quer gerar. Basicamente, são informadas as colunas de entrada (input) e saída (output), ou seja, citando o exemplo de um SELECT os campos de entrada são os que farão parte de sua seção WHERE enquanto que os campos de saída são os que deverão ser retornados. Ilustrando este exemplo, a **Figura 14** mostra a janela de configuração para a criação de um Statement SELECT, em que o código SQL resultante é exibido a seguir:

```
SELECT
        t.FIRST_NAME,
        t.LAST_NAME,
        t.SALARY
FROM EMPLOYEE t
WHERE EMP_NO = /* VALOR */
```

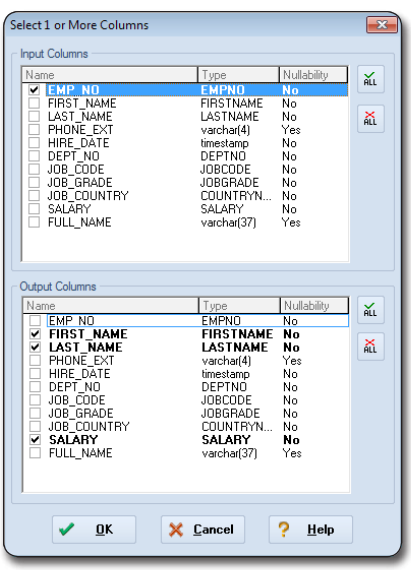


Figura 14. Configurando colunas de entrada e saída para a geração de um Select

A outra forma de se efetuar tal processo é através da opção Code Generation Facility, presente no menu Tools do Rapid SQL e que opera de maneira muito semelhante à opção anterior. Seu

diferencial fica por conta da possibilidade de direcionar o código SQL produzido a arquivos externos, além de possibilitar a geração automática de código para a criação de outros tipos de objetos e não somente Stored Procedures.

Query Builder

Escrever instruções SQL complexas pode despender um tempo precioso durante o desenvolvimento e que sofre um aumento gradativo conforme for o tamanho e complexidade da estrutura do banco de dados. Para situações como esta o Rapid SQL tem disponível um importante recurso que permite com que se construam tais instruções SQL de forma totalmente visual, somente com a ação do mouse. Este tipo de funcionalidade pode, inclusive, facilitar o trabalho de outro tipo de profissional, aquele menos experiente e com pouco conhecimento da linguagem SQL.

O Query Builder (também chamado de Visual Query Builder) é disponibilizado no menu Tools do IDE e sua ambientação é mostrada na **Figura 15**. Toda sua "magia" ocorre por via de suas duas guias principais: Tables/Views e DML. Na primeira é onde estão listados os objetos do banco de dados que poderão ser arrastados para sua área principal e assim dar início à montagem da instrução SQL. Já na guia DML é onde poderão ser customizados os blocos constituintes da instrução, como From, Join, Where, Order By, Group By, etc. Finalmente, mediante o resultado obtido pela montagem visual do Query Builder é mostrada a instrução SQL formada no rodapé da janela, na guia SQL. A qualquer momento este código pode ser testado ou executado através de dois botões de comandos presentes na sua barra de ferramentas, denominados Check Syntax e Execute, respectivamente. Vale lembrar que dependendo da quantidade de dados retornados pela instrução que está sendo montada, sua execução pode ser um tanto quanto demorada. Sendo assim, caso seja de interesse somente a verificação pela integridade do código que está sendo montado, o indicado é o uso da opção Check Syntax que apenas faz uma conferência do código, sem executá-lo efetivamente.

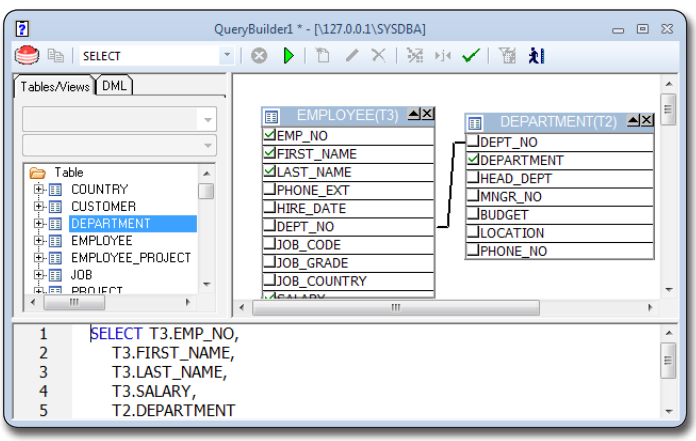


Figura 15. Query Builder

Nota do DevMan

Em banco de dados, DML (Data Manipulation Language) é uma linguagem de manipulação de dados, tais como consulta (Select), inclusão (Insert), alteração (Update) e exclusão (Delete) que, assim como a DDL (Data Definition Language), é determinada como um subconjunto da linguagem SQL.

Conclusão

A intenção principal do artigo foi a de expor de maneira simples e objetiva alguns dos principais recursos da ferramenta Rapid SQL da Embarcadero que, assim como o Delphi, faz parte da família de produtos XE2. Como consequência visa-se apresentar ao desenvolvedor mais uma opção de trabalho no que diz respeito à manutenção de bancos de dados. Isto porque, tradicionalmente, é comum este tipo de profissional ter que além de programar, cuidar da base de dados relacionada ao projeto.

Vale ressaltar que o Rapid SQL é uma ferramenta muito poderosa para o trabalho com banco de dados e suas funcionalidades vão muito além do apresentado e seria necessário mais que uma revista por completo para abordar cada uma. Portanto, a dica para aqueles que se interessaram pela ferramenta é tomar este artigo como ponto de partida e ir se aprofundando nos detalhes relativos às necessidades de sua realidade de trabalho, tendo como apoio os materiais oferecidos pela própria fabricante na página do produto. Agradeço a todos e até a próxima.

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Blog: prismsilver.blogspot.com
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 6 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas.

Embarcadero Rapid SQL XE2 – Página oficial do produto
http://embarcadero.com/br/products/rapid-sql-xe2

Firebird – Página oficial do produto
http://www.firebirdsql.org

Firebird – Driver ODBC
http://www.firebirdsql.org/en/odbc-driver

Embarcadero Interbase XE - Página official do produto
http://embarcadero.com/br/products/interbase

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

FireMonkey e NFe 2.0

Uma solução passo a passo – Pa

Resumo DevMan

De que trata o artigo:

Esta série de artigosapresenta uma aplicação para a emissão, assinatura, validação, envio, consulta, entre outros, da Nota Fiscal Eletrônica, destinada à recente plataforma do Delphi XE 2, o FireMonkey. Esta terceira parte tem por objetivo demonstrar todos os passos necessários para enviar e autorizar uma NF-e, consumindo os WebServices da SEFAZ.

Em que situação o tema é útil:

Permitir a inclusão de um módulo de NF-e em uma aplicação FireMonkey existente de uma maneira nativa, fazendo com a mesma não necessite recorrer a integração com outras aplicações para oferecer esta funcionalidade, bem como apresentar o funcionamento de alguns dos WebServices da SEFAZ.

FireMonkey e NFe 2.0:

A NF-e é um recurso importante que deve ser disponibilizado pelas aplicações comerciais e industriais para não perderem mercado, uma vez que grande parte das empresas éobrigada a emitir a Nota Fiscal Eletrônica. Esta terceira parte apresenta uma introdução ao conceito de WebService, um recurso muito adotado atualmente para automatizar a troca de informações na Internet, tendo como foco os WebServices disponibilizados pela Secretaria da Fazenda, bem como apresentar todos os processos envolvidos para que a NF-e possa ser enviada e autorizada, garantindo assim sua validade jurídica.

Nos artigos anteriores deste minicurso foram abordadas as diversas etapas para o início da emissão da NF-e. A primeira parte em suma, teve seu foco direcionado na elaboração da Nota Fiscal e das funções necessárias para gerar as informações, levando em consideração os requisitos da SEFAZ e a importação de umSchema baseado em XML (ExtensibleMarkupLanguage) através do recurso XML Data Binding, no qual o Delphi gera uma Unit contendo as interfaces e métodos obtidos de acordo com o Schema definido. Na segunda parte foi abordada a utilização do certificado digital, iniciando uma introdução à CAPICOM, a biblioteca que permite a utilização de recursos de segurança em aplicações desenvolvidas em Delphi e outras ferramentas para o ambiente Windows, para assinar os documentos XML utilizando um certificado digital, bem como a MSXML, que é uma biblioteca que permite a utilização de DOM (DocumentObjectModel), um recurso muito importante e que facilita a manipulação dos documentos no padrão XML, para auxiliar na validação da NF-e. É importante salientar que a obtenção do certificado digital privado por parte da empresa é uma obrigatoriedade para a emissão da NF-e, e que o mesmo possui um tempo de validade determinado devendo ser renovado assim que é expirado. Contudo, também não se deve esquecer que existe a necessidade da instalação dos certificados públicos que pertencem à cadeia de certificação da hierarquia ICP-Brasil, para serem utilizados em diversos momentos na aplicação de NF-e.

Outro fator que deve ser observadoe acompanhado de perto pelos utilizadores e desenvolvedores das aplicações de NF-e são as atualizações que ocorrem tanto nos XML Schemas quanto nos certificados. Isto é importante pois quando as correções necessárias não são efetuadas, as aplicações de Nota Fiscal Eletrônica correm o risco de não conseguirem consumir os WebServices da Secretaria da Fazenda, impossibilitando a emissão da nota.

Após a Nota Fiscal ser concebida, assinada e validada, a mesma necessita ser enviada para um determinado WebService da Secretaria da Fazenda para que ela possa ser considerada válida. O WebService é um recurso padronizado que é disponibilizado em uma rede como a Internet, por uma determinada organização, com a finalidade de prover vários recursos através da Internet e esses mesmos podem ser gratuitos ou pagos.

O WebService é implementado para ser autossuficiente e receber as requisições, processar e efetuar suas eventuais respostas de maneira automática, possibilitando assim uma consulta praticamente instantânea sobre determinado serviço. Contudo, cada WebSevice possui seus próprios métodos e forma de funcionamento interno, sendo assim o desenvolvedor precisa conhecer suas funcionalidades para que o mesmo possa consumir seus recursos, por este motivo existe o WSDL (WebService DescriptionLanguage), que é um documento baseado no padrão XML que descreve os recursos do WebService em questão, demonstrando seus métodos e funções.

Além dos softwares e ferramentas necessários para a elaboração e disponibilização, os WebServices também são baseados nos protocolos HTTP (HyperTextTransferProtocol) e SOAP (SimpleObject Access Protocol). O HTTP é o protocolo padrão de transporte no cenário da Web e o SOAP é um protocolo baseado em XML padroni-

Giuliano Scombatti Pinto

zado, cuja finalidade é permitir que os serviços fornecidos pelos WebServices possam ser chamados e posteriormente seu retorno seja recebido, bem como apresentar os dados necessários para que este processo possa ser efetuado através das mensagens. Por ser um padrão estabelecido pela W3C e baseado simplesmente em texto, uma mensagem padrãoem SOAP é compatível com os mais variados ambientes e ferramentas de desenvolvimento presentes no mercado, sendo que tal protocolo possui diversas versões. A versão do protocolo SOAP adotada pela SEFAZ para a NF-e 2.0 é a 1.2. Entretanto, muitos WebServices, como é o caso dos disponibilizados pela Secretaria da Fazenda, podem utilizar uma implementação de segurança para seu funcionamento, que normalmente é baseada em SSL (Secure Socket Layer) sobre o protocolo HTTP, que recebe a denominação de HTTPS (Hyper-TextTransferProtocolSecure). A utilização deste recurso permite que usuários sejam autenticados, bem como mantém a integridade das informações oferecendo privacidade, tendo apenas a desvantagem de ser um pouco mais lenta que a aplicação convencional baseada somente em HTTP.

Outro fator que merece destaque é com relação aos recursos envolvidos para que um WebService possa estar disponível, ou seja, o mesmo depende de outros Softwares, da Internet e consequentemente de Hardware, sendo assim não há como garantir que um WebService esteja sempre em funcionamento. Este é um dos motivos de existirem outras modalidades na Emissão da NF-e, que devem ser adotadas como contingência, tais como o SCAN (Sistema de Contingência do Ambiente Nacional) e o DPEC (Declaração Prévia de Emissão em Contingência).

Contudo, ao consumir os WebServices da SEFAZ, deve-se definir em qual ambiente o mesmo será executado, ou seja, se em homologação (testes) ou produção, uma vez que os endereços dos WebServices são diferentes. Todo o desenvolvimento deste projeto deve ser realizado no ambiente de homologação para evitar quaisquer problemas. Sendo assim, uma vez que tudo esteja operando satisfatoriamente no ambiente de homologação, basta substituir os endereços (URL – UniformResourceLocator) dos WebServices deste ambiente para o ambiente de produção, dando início a validade das notas emitidas pela aplicação NF-e.

Etapas do processo de envio e autorização da NF-e em modo Normal

A Internet possibilita diversas alternativas para a troca de informações e o processo de Nota Fiscal Eletrônica é mais um recurso que utiliza esta base. Uma vez que a NF-e está concluída, o envio da mesma deve ser efetuado para o WebService da Receita. Um detalhe que deve ser observado é que o processo de envio e autorização da NF-e é constituído por algumas etapas. O desenvolvedor também pode elaborar a aplicação de NF-e para enviar várias notas de uma só vez, dentro de um mesmo lote.

Nota do DevMan

O XML (ExtensibleMarkupLanguage) é um documento baseado em Textoutilizado para o armazenamento de vários conjuntos de dados. Devido a sua facilidade de elaboração e tamanho é um dos padrões mais adotados para a troca de informações na Internet, uma vez que o mesmo não possui uma limitação para as informações. O XML tem sua estrutura baseada em elementos ou nós que seguem uma determinada hierarquia.

Nota do DevMan

A W3C (World Wide Web Consortium) é um consórcio internacional que estabelece diversos padrões para a Internet, constituído por diversas entidades. Os padrões estabelecidos pela W3C são aplicados em diversos ambientes, como é o caso da Internet, possibilitando a troca de informações entre plataformas diferentes.

Nota do DevMan

URL (UniformResourseLocator) é a denominação utilizada para definir o endereço de determinado recurso em uma rede. Ela também pode definir qual deve ser o protocolo de comunicação para o acesso.

O primeiro passo deste processo de envio e autorização de NF-e é transmitir a Nota Fiscal versão 2.0 para o WebService padrão de recepção da SEFAZ (NfeRecepcao2), lembrando que os certificados digitais (privado e público) são necessários em todas as fases deste processo para a autenticação. Após o consumo deste WebService é obtida uma mensagem de retorno SOAP contendo diversas informações no padrão XML, entretanto os dados relevantes estão presentes nas TAGs <xMotivo>, que deverá conter a mensagem "Lote recebido com sucesso" , e <nRec>, que armazena o número de recibo do lote.Evidentemente se houver algum erro durante todo este processo o retorno obtido será uma mensagem contendo informações sobre o mesmo. Contudo, quando este processo é concluído satisfatoriamente, significa que o WebService da receita recebeu a NF-e, porém ainda não implica que a mesma está autorizada para o uso. Por esse motivo

convém à aplicação de NF-e armazenar o arquivo de retorno para que possa ser utilizado futuramente. Após alguns segundos, a aplicação cliente de NF-e pode efetuar o consumo de um segundo WebService da SEFAZ (NfeRetRecepcao2) para a consulta do status da NF-e transferida, enviando em conjunto com a mensagem de requisição SOAP, o número do recibo que foi obtido durante o processo de envio (<nRec>). Se a NF-e emitida está de acordo com os requisitos da SEFAZ, através da consulta deste WebService será possível obterum documento de retorno também sobre o padrão SOAP contendo diversas informações das quais se destacam as TAGs<xMotivo>, que apresenta a mensagem "Autorizado o uso da NF-e", indicando que a NF-e foi aprovada, bem como a TAG <nProt>, que traz consigo o número do protocolo da mesma, evidentemente se houver algum imprevisto ou problema detectado na NF-e, a mensagem de retorno conterá as informações relacionadas a esta ocorrência. Efetuadas estas etapas com sucesso, o próximo passo é anexar as informações obtidas em um único documento XML (procNFe) constituído pela NF-e validada que foi enviada e as informações retornadas pela última consulta (infProt) para então prosseguir para o passo final do processo geral da NF-e que é a impressão da DANFE, o Documento Auxiliar da Nota Fiscal Eletrônica. Este deve ser utilizado para o transporte das mercadorias e futuras consultas, e o envio do documento XML para o destinatário da nota através de e-mail ou outro recurso, uma vez que este passo é obrigatório.

Nota do DevMan

A denominação TAG no ambiente computacional é utilizada para definir marcas de início (<nomeTAG>) e fim (</nomeTAG>) de determinada informação em linguagens de marcação, como é o caso do XML, permitindo assim uma organização dosdados de acordo com palavras-chave.

Envio da NF-e e resposta na prática

Após ter uma visão geral do processo de envio da NF-e, o próximo passo é a implementação do mesmo. Para tanto, basta abrir o mesmo projeto que foi utilizado nos artigos anteriores. Um ponto importante que deve ser evidenciado neste momento é que existem várias maneiras de efetuar todo este processo, e o caminho que será seguido vai depender da necessidade do projeto. Entre eles, o desenvolvedor pode efetuar o XML Data Binding no XMLSchemaconsReciNFe_v2.00.xsd para extrair as interfaces e poder manipular diversas informações de retorno das notas utilizando um objeto TXMLDocument por exemplo, que representa um documento XML e oferece diversas funcionalidades.Também pode-se ser mais direto, manipulando as informações obtidas da mesma maneira como manipula uma String, fazendo uso das funções Pos() e StringReplace(), entre outros caminhos que podem ser utilizados. Para este projeto, foi efetuado o XML Data Binding (File>New>Other>XML>XML Data Binding) no Schema XML mencionado anteriormente (consReciNFe_v2.00.xsd) para uma melhor manipulação dos retornos obtidos pelas consultas e demonstrar uma maneira de utilizar o mesmo. Também foram criados alguns elementos, tais como um objetodo tipo THTTPReqResp, cujo objetivo é permitir o consumo de WebServices através do protocolo HTTP ou HTTPS, euma mensagem no padrão SOAP diretamente em uma variável String, uma vez que o funcionamento do WebService seja conhecido.

Para dar início ao desenvolvimento desta etapa, foi criada uma função para facilitar a obtenção de determinado valor presente em uma TAG, que futuramente será utilizada para a filtragem das informações desejadas. A **Listagem 1** apresenta a função getConteudoTAG() que tem o propósito de obter o valor presente em uma TAG e será utilizada neste exemplo para obter algumas informações dos arquivos de retorno dos WebServices, assim como ler outros dados.

Listagem 1. Função para obter o valor de uma TAG

```
function TFormMovEnvNotaFiscal.getConteudoTAG(ArquivoXML,
NomeTAG, pTAG: AnsiString): AnsiString;
var
  PosIni, PospTAG: Integer;
begin
  if Trim(pTAG) <> '' then
  begin
    PospTAG := Pos('<'+ pTAG, ArquivoXML);
ArquivoXML := Copy(ArquivoXML, PospTAG, Pos('</'+ pTAG, ArquivoXML) -
PospTAG);
end;
  PosIni := Pos('<'+ NomeTag, ArquivoXML) + Length(NomeTag) + 2;
  Result := Copy(ArquivoXML, PosIni, Pos('</'+ NomeTag, ArquivoXML) - PosIni);
end;
```

Esta função tem como parâmetros as variáveis ArquivoXML, utilizada para receber a string do XML, aNomeTAG, que recebe o nome da TAG pela qual se deseja obter o valor, e a pTAG, cujo o objetivo é especificar em qual nó "pai" deverá ser efetuada a busca, uma vez que determinados arquivos de retorno podem conterTAGs iguais em diferentes elementos. Sendo assim, é verificado se foi especificado algum nó pai (pTAG) e caso tenha sido, a função Pos() é utilizada para efetuar uma pré-filtragem nas informações, preservando somente o que está contido nesta TAG.Na sequência, a função Pos() é utilizada novamente para identificar qual a primeira ocorrência da TAG passada como parâmetro (NomeTAG), somando o tamanho (Lenght) da mesma com o valor (2) dos delimitadores da TAG (<>). Para finalizar, o resultado da função é obtido através da função Copy(), utilizada para copiar ou recortar uma String, tendo como parâmetros a string XML, a posição Inicial, e a posição da informação que é obtida através da posição de início da TAG de encerramento (</ >) menos a Posição Inicial.Como pôde ser observado, esta função é muito útil para ser aplicada neste projeto, pois permite a obtenção do valor presente na primeira ocorrência de determinada TAG, oferecendo a opção de filtrar também por elemento pai.

Com esta função criada, o próximo passo é elaborar outra para controlar o número do lote que será enviado. Mesmo que seja enviada apenas uma NF-eeste número deve ser gerado e futuramente anexado na mensagem SOAP, onde convém criar um campo incremental no banco de dados para controlar tal valor. Entretanto neste exemplo, para não criar vínculos com um banco de dados, é criada uma simples função que gera o Id do Lote de acordo com o número da NF-e obtido através do XML, lembrando que este número deve ser sequencial e que esta função é destinada ao envio de apenas uma NF-e por vez, devendo a mesma ser alterada para o envio de várias notas fiscais em um mesmo lotese houver necessidade no projeto.A **Listagem 2** apresenta a função utilizada para gerar o número identificador para o Lote de acordo com a NF-e.

Listagem 2. Função para gerar o Id do Lote para envio de uma NF-e

```
function TFormMovEnvNotaFiscal.gerarIdLote(ArquivoXML: AnsiString):
AnsiString;
var
  PosId: Integer;
  IdLote: AnsiString;
begin
PosId := Pos('Id=', ArquivoXML);
IdLote := Copy(ArquivoXML, PosId + 35, 6);
IdLote := FormatDateTime('yyyymmdd', Date) + IdLote;
Result := IdLote;
end;
```

A função Pos() é utilizada na variável ArquivoXML para obter a posição do atributo Id, para que seja utilizada a função Copy(),obtendo assim o número da Nota que está presente em tal atributo. Na sequência, a função FormatDateTime(), utilizada para formatar uma data e/ou hora retornando um valor em string de acordo com o parâmetro informado, é utilizada para criar a sequência ano, mês e dia, que é adicionada antes do número da nota, lembrando que esta é apenas uma forma de gerar o Id do Lote que evidentemente deve ser implementado de acordo com o projeto ou desejo do desenvolvedor. Pode haver rejeição caso o número do lote não ser informado ou ser informado de uma maneira incorreta.

Criadas as funções auxiliares descritas anteriormente, será elaborada a função de envio da NF-e. Neste momento antes de executar o envio da NF-e, o desenvolvedor pode efetuar uma consulta para verificar o Status do Serviço, obtendo assim a informação se o mesmo está em operação através da função ConsultarStatusServico(), que foi elaborada no segundo artigo com o intuito de garantir que a NF-e pode ser enviada, contudo este passo é opcional. Para utilizar esta função, a Unit consReciNFe_v200 obtida através do processo de XML Data Binding efetuado anteriormente, deve ser adicionada na seção Uses da atual Unit. A **Listagem3** demonstra a função elaborada para a transferência da NF-e 2.0 para o WebService de recepção da SEFAZ (NFeRecepcao2), recebendo em seguida o retorno do mesmo.

Primeiramente na declaração de variáveis existe um tipo incomum que é o IXMLTRetEnviNFe. Este tipo é uma interface derivada de uma IXMLNode, que por sua vez é utilizada para representar um nó no documento XML, sendo tal tipo obtido através da Unit consReciNFe_v200.pas criada anteriormente. Ele pode ser utilizado para armazenar e manipular a resposta obtida no envio da NF-e.

Na sequência do código é criado um objeto do tipo THTTPReq Resp. Este objeto pode ser utilizado para efetuar uma solicitação a um WebService e obter a mensagem de resposta do mesmo através do protocolo HTTP ou HTTPS. A seguir, são definidas algumas propriedades do objeto HTTPReqResp. Algumas destas propriedades são a base para o funcionamento deste objeto, sendo elas: a URL contendo o endereço do WebService, que neste caso é de homologação mas poderia ser simplesmente substituído pelo de produção, e a SoapAction que define o cabeçalho da mensagem para a efetuar a chamada a um método do WebService. Posteriormente é criada a Lista do tipoTStringList, uma lista de strings que armazenará a mensagem SOAP. No próximo passo é definida a mensagem SOAP que será enviada para o WebService anexando também o arquivo XML da NF-e, uma vez que o funcionamento e as exigências do WebService sejam conhecidos, a mensagem pode ser criada diretamente em texto. A propriedade Text da variável Lista recebe a mensagem SOAP para possibilitar a utilização do HTTPReqRespem conjunto com a variável MemoriaStream, do tipo TMemoryStream. O tipo TMemoryStream é muito útil quando informações devem ser manipuladas na memória, possibilitando também que a mesma seja exportada. Sendo assim, a variável TextoStream, do tipo TStringStream, copia o conteúdo da variável MemoriaStream através do método CopyFrom() para possibilitar assim uma melhor manipulação com relação a strings. A variável Resultado, por sua vez, recebe a string do XML de retorno obtido através da TextoStream. Neste momento o objeto XMLDoc, do tipo TXMLDocument, é criado e o objeto RetEnviNFerecebe o Binding do Documento XML através da função GetDocBinding(). Esta função é utilizada para especificar qual é a classe para a implementação que será utilizada, retornando a interface do elemento especificado bem como o NameSpaceURI do documento XML. Sendo assim, o objeto RetEnviNFe tem seus nós filhos atribuídos de acordo com a resposta obtida que foi armazenada na variável Resultado, onde o filtro para cada TAG é efetuado através da função criada anteriormente, getConteudoTAG(). Para concluir, o arquivo XML é gravado no disco através da função SaveToFile() do TXML Document e o valor retornado por esta função (Result) é composto apenas pelas principias informações obtidas pelo retorno.

Nota do DevMan

O Namespace URI é utilizado para evitar possíveis conflitos em nomes de elementos em documentos XML, atribuindo um nome exclusivo aos mesmos. Ele também é um padrão que foi elaborado pela W3C e deve ser definido no atributo xmlns na TAG de um elemento.

Listagem 3. Função para o envio da NF-e

```
function TFormMovEnvNotaFiscal.EnviarNFe(ArquivoXML: AnsiString): AnsiString;
var
  MemoriaStream: TMemoryStream;
  TextoStream: TStringStream;
  MsgSOAP, Resultado: AnsiString;
  Lista: TStringList;
  HTTPReqResp: THTTPReqResp;
  IdLote, ChaveNFe: String;
  PosId: Integer;
  RetEnviNFe: IXMLTRetEnviNFe;
  XMLDoc: TXMLDocument;
begin
  HTTPReqResp := THTTPReqResp.Create(FormMovEnvNotaFiscal);
with HTTPReqResp do
begin
    InvokeOptions := [soIgnoreInvalidCerts,soNoSOAPActionHeader,
              soPickFirstClientCertificate,soAutoCheckAccessPointViaUDDI];
    URL := 'https://homologacao.nfe.fazenda.sp.gov.br/nfeweb/services/
    NfeRecepcao2.asmx';
    UseUTF8InHeader := True;
    SoapAction := 'http://www.portalfiscal.inf.br/nfe/wsdl/NfeRecepcao2';
  end;
  Lista := TStringList.Create;
  PosId := Pos('Id=', ArquivoXml) + 7;
  ChaveNFe := Copy(ArquivoXML,PosId, 44);
  IdLote := gerarIdLote(ArquivoXML);
  MsgSOAP := '<?xml version="1.0" encoding="utf-8"?>'+
        '<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"'+
        'xmlns:xsd="http://www.w3.org/2001/XMLSchema"'+
        'xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">'+
        '<soap12:Header>'+
       '<nfeCabecMsg xmlns="http://www.portalfiscal.inf.br/nfe/wsdl/NfeRecepcao2">
      '+'<cUF>35</cUF>'+
        '<versaoDados>2.00</versaoDados>'+
        '</nfeCabecMsg>'+
        '</soap12:Header>'+
        '<soap12:Body>'+
       '<nfeDadosMsg xmlns="http://www.portalfiscal.inf.br/nfe/wsdl/NfeRecepcao2">
       '+'<enviNFe xmlns="http://www.portalfiscal.inf.br/nfe" versao="2.00">
       '+'<idLote>'+ IdLote +'</idLote>'+
          ArquivoXml +
          '</enviNFe>'+
          '</nfeDadosMsg>'+
          '</soap12:Body>'+
          '</soap12:Envelope>';
  try
    Lista.Text := MsgSOAP;
    MemoriaStream := TMemoryStream.Create;
    HTTPReqResp.Execute(Lista.Text, MemoriaStream);
    TextoStream := TStringStream.Create('');
    TextoStream.CopyFrom(MemoriaStream, 0);
Resultado := UTF8Decode(TextoStream.DataString);
    XMLDoc := TXMLDocument.Create(nil);
XMLDoc.Active := True;
     RetEnviNFe := XMLDoc.GetDocBinding('retEnviNFe', TXMLTRetEnviNFe,
    'http://www.portalfiscal.inf.br/nfe') as IXMLTRetEnviNFe;
with RetEnviNFe do
begin
      Versao := '2.00';
TpAmb := getConteudoTAG(Resultado, 'tpAmb', '');
       verAplic := getConteudoTAG(Resultado, 'verAplic', '');
       cStat := getConteudoTAG(Resultado, 'cStat', '');
       xMotivo := getConteudoTAG(Resultado, 'xMotivo', '');
       cUF := getConteudoTAG(Resultado, 'cUF', '');
      dhRecbto := getConteudoTAG(Resultado, 'dhRecbto', '');
       InfRec.NRec := getConteudoTAG(Resultado, 'nRec', '');
       InfRec.TMed := getConteudoTAG(Resultado, 'tMed', '');
end;
    XMLDoc.SaveToFile('C:\FMXNFE\NfeRecibos\' + ChaveNFe + '-rec.xml');
    Resultado := 'Resposta: '+ RetEnviNFe.XMotivo +' / '+
           'Recibo: '+ RetEnviNFe.InfRec.NRec;
XMLDoc.Active := False;
    Result := Resultado;
finally
    FreeAndNil(HTTPReqResp);
    FreeAndNil(MemoriaStream);
    FreeAndNil(TextoStream);
    FreeAndNil(Lista);
end;
end;
```

Enviada a NF-e satisfatoriamente, é preciso consultar a mesma para verificar se ela foi autorizada ou caso contrário, obter o motivo pela qual a mesma foi rejeitada, lembrando que convém aguardar alguns segundos após o envio da NF-e para efetuar este processo, e dependendo do funcionamento da aplicação, o desenvolvedor pode inserir um objeto TTimer para efetuar consultas periódicas para a obtenção do retorno do Lote, desativando o mesmo assim que o resultado da autorização é obtido. A **Listagem 4** apresenta a função para a consulta da NF-e baseada no número do recibo (<nRec>) que pode ser obtido através do arquivo de resposta do envio que foi gravado anteriormente no disco.

Esta função também apresenta algumas particularidades na seção de declarações de variáveis, como é o caso dos tipos IXMLTRetConsReciNFe, IXMLInfProt e IXMLTProtNFeList. Ambos os objetos se encontram na Unit consReciNFe_v200.pas, sendo os dois primeiros derivados da interface IXMLNode, utilizada para representar nós e o último de uma interface IXMLNodeCollection, utilizada para representar um conjunto de nós filhos.No início do código da função também é criado um objeto do tipo THTTPReqResp para efetuar a requisição ao WebService de consulta (NfeRetRecepcao2). Também são definidas as propriedades do mesmo para o acesso. Posteriormente é criada a Lista do tipo TStringList, uma lista de strings, que armazenará a mensagem SOAP, sendo também obtido o número do recibo da NF-e enviada através do documento XML de retorno com a função getConteudoTAG(). No próximo passo é definida a mensagem SOAP que será enviada para o Webservice contendo a variável que recebeu o número do recibo para a consulta,NRecibo. A propriedade Text da variável Lista recebe a mensagem SOAP possibilitando a futura utilização do HTTPReqResp e a variável MemoriaStream, do tipo TMemoryStream. Em seguida, a variável MemoriaStream tem seu conteúdo copiado pela variável TextoStream, do tipo TString Stream. É atribuída a string do XML de retorno à variável Resultado. Em seguida, o objeto XMLDoc (TXMLDocument) é criado e o objeto RetConsReciNFe (IXMLTRetConsReciNFe) recebe o Binding do Documento XML, lembrando que a interface IXMLTRetConsReci NFe é utilizada para a manipulação das informações de retorno deste tipo de consulta. Sendo assim, o objeto RetConsReciNFe tem seus nós filhos atribuídos em conjunto com a variável Resultado e a função getConteudoTAG(), bem como são atribuídas as interfaces ProtNFeList e InfProt, para manipular as informações detalhadas da

Listagem 4. Função para a consulta da NF-e

```
function TFormMovEnvNotaFiscal.ConsultarNRecNFe(ArquivoXml: AnsiString):
AnsiString;
var
  MemoriaStream: TMemoryStream;
  TextoStream: TStringStream;
  MsgSOAP, Resultado: AnsiString;
  Lista: TStringList;
  HTTPReqResp: THTTPReqResp;
  NRecibo: String;
  RetConsReciNFe: IXMLTRetConsReciNFe;
  ProtNFeList: IXMLTProtNFeList;
  InfProt: IXMLInfProt;
  XMLDoc: TXMLDocument;
begin
  HTTPReqResp := THTTPReqResp.Create(FormMovEnvNotaFiscal);
  with HTTPReqResp do
   begin
    InvokeOptions := [soIgnoreInvalidCerts,soNoSOAPActionHeader,
                soPickFirstClientCertificate,soAutoCheckAccessPointViaUDDI];
    URL := 'https://homologacao.nfe.fazenda.sp.gov.br/nfeweb/services/
    NfeRetRecepcao2.asmx';
    UseUTF8InHeader := True;
    SoapAction := 'http://www.portalfiscal.inf.br/nfe/wsdl/NfeRetRecepcao2';
   end;
  Lista := TStringList.Create;
  NRecibo := getConteudoTAG(ArquivoXML, 'nRec');
  MsgSOAP := '<?xml version="1.0" encoding="utf-8"?>'+
        '<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"'+
        'xmlns:xsd="http://www.w3.org/2001/XMLSchema"'+
        'xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">'+
        '<soap12:Header>'+
        '<nfeCabecMsg xmlns="http://www.portalfiscal.inf.br/nfe/wsdl/
          NfeRetRecepcao2">'+
        '<cUF>35</cUF>'+
        '<versaoDados>2.00</versaoDados>'+
        '</nfeCabecMsg>'+
        '</soap12:Header>'+
        '<soap12:Body>'+
        '<nfeDadosMsg xmlns="http://www.portalfiscal.inf.br/nfe/wsdl/
          NfeRetRecepcao2">'+
        '<consReciNFe xmlns="http://www.portalfiscal.inf.br/nfe" versao="2.00">' +
        '<tpAmb>2</tpAmb>'+
        '<nRec>'+ NRecibo +'</nRec>'+
        '</consReciNFe>'+
        '</nfeDadosMsg>'+
        '</soap12:Body>'+
        '</soap12:Envelope>';
  try
    Lista.Text := MsgSOAP;
    MemoriaStream := TMemoryStream.Create;
    HTTPReqResp.Execute(Lista.Text, MemoriaStream);
    TextoStream := TStringStream.Create('');
    TextoStream.CopyFrom(MemoriaStream, 0);
    Resultado := UTF8Decode(TextoStream.DataString);
    XMLDoc := TXMLDocument.Create(nil);
    XMLDoc.Active := True;
    RetConsReciNFe := XMLDoc.GetDocBinding('retConsReciNFe', TXMLTRetConsReci
    NFe, 'http://www.portalfiscal.inf.br/nfe') as IXMLTRetConsReciNFe;
  with RetConsReciNFe do
  begin
      Versao := '2.00';
      TpAmb := getConteudoTAG(Resultado, 'tpAmb');
      verAplic := getConteudoTAG(Resultado, 'verAplic');
      NRec := getConteudoTAG(Resultado, 'nRec');
      cStat := getConteudoTAG(Resultado, 'cStat');
      xMotivo := getConteudoTAG(Resultado, 'xMotivo');
      cUF := getConteudoTAG(Resultado, 'cUF');
  end;
    ProtNFeList := RetConsReciNFe.ProtNFe;
    ProtNFeList.Add;
    ProtNFeList.Items[0].Versao := '2.00';
    InfProt := ProtNFeList.Items[0].InfProt;
  with InfProt do
  begin
      TpAmb := getConteudoTAG(Resultado, 'tpAmb');
      verAplic := getConteudoTAG(Resultado, 'verAplic');
      chNFe := getConteudoTAG(Resultado, 'chNFe');
      dhRecbto := getConteudoTAG(Resultado, 'dhRecbto');
      nProt := getConteudoTAG(Resultado, 'nProt');
      digVal := getConteudoTAG(Resultado, 'digVal');
      cStat := getConteudoTAG(Resultado, 'cStat');
      xMotivo := getConteudoTAG(Resultado, 'xMotivo'infProt');
  end;
    XMLDoc.SaveToFile('C:\FMXNFE\NfeProtocolos\' + InfProt.ChNFe + '-prot.xml');
    Resultado := 'Resposta: '+ InfProt.XMotivo +' / '+
          'Protocolo: '+ InfProt.NProt;
    XMLDoc.Active := False;
  Result := Resultado;
  finally
    FreeAndNil(HTTPReqResp);
    FreeAndNil(MemoriaStream);
    FreeAndNil(TextoStream);
    FreeAndNil(Lista);
  end;
end;
```

consulta. Para concluir, o arquivo XML é gravado em um diretório no disco e a função retorna as principias informações obtidas pelo resultado desta consulta.

Neste momento, após a obtenção de sucesso nos retornos, o processo de consumo de WebServices é encerrado para a etapa de envio e autorização da NF-e restando apenas anexar a NF-e validada e as informações do protocolo em um único arquivo para a distribuição. A **Listagem 5** demonstra uma função para anexar a NF-e com seu protocolo para que a mesma possa ser distribuída, encerrando assim a última etapa deste processo.

No primeiro momento deste código é obtida a posição do atributo Id com relação ao documento XML da NF-e validada para a obtenção da chave da NF-e logo em seguida. Posteriormente são criados dois documentos XML (TXMLDocument), um para carregar a string da NF-e através da função LoadFromXML(), e o outro para carregar o arquivo XML que contém as informações de autorização através da função LoadFromFile(). Efetuado este processo os objetos do tipo IXMLNode, NoPrincipalNFe e NoProtocolo, recebem os nós que devem ser anexados para a elaboração do arquivo XML protNFe, que são respectivamente o nó principal da NF-e validada e o nó protNFe do arquivo que contém o protocolo de autorização. Em seguida é criado o documento XML (DocFinal) que será utilizado para a elaboração do documento XML. Tal documento também recebe as definições de alguns atributos, bem como é definido o elemento raiz do mesmo. Em seguida, são adicionados os nós NoPrincipalNFe e NoProtocolo através da função ChildNodes.Add(). Para concluir, o documento XML (DocFinal) é gravado em um diretório do disco rígido, completando esta fase da aplicação NF-e.

Listagem 5. Função para anexar a NF-e e seu protocolo de autorização

```
function TFormMovEnvNotaFiscal.FinalizarNFe(ArquivoXML: AnsiString): AnsiString;
var
  PosId: Integer;
  ChaveNFe: String;
  DocNFe, DocProtocolo, DocFinal: IXMLDocument;
  NoPrincipalNFe, NoProtocolo, DocElem: IXMLNode;
  begin
  PosId := Pos('Id=', ArquivoXml) + 7;
  ChaveNFe := Copy(ArquivoXML,PosId, 44);
  DocNFe := TXMLDocument.Create(nil);
  DocProtocolo := TXMLDocument.Create(nil);
try
    DocNFe.LoadFromXML(ArquivoXML);
    DocProtocolo.LoadFromFile('C:\FMXNFE\NfeProtocolos\'+ ChaveNFe +'-prot.xml');
    NoPrincipalNFe := DocNFe.DocumentElement;
    NoProtocolo := DocProtocolo.DocumentElement.ChildNodes.FindNode('protNFe');
     DocFinal := TXMLDocument.Create(nil);
with DocFinal do
 begin
     Active := True;
     AddChild('nfeProc','http://www.portalfiscal.inf.br/nfe');
     DocElem := DocumentElement;
with DocElem do
 begin
        Attributes['versao'] := '2.00';
        ChildNodes.Add(NoPrincipalNFe);
        ChildNodes.Add(NoProtocolo);
 end;
     Version := '1.0';
     Encoding := 'UTF-8';
 end;
 DocFinal.SaveToFile('C:\FMXNFE\NfeFinalizadas\'+ ChaveNFe +'.xml');
 Result := ChaveNFe;
 except
 on e: Exception do
 ShowMessage('Houve um erro: '+ e.Message);
 end;
end;
```

Conclusão

O desenvolvimento de um módulo para a utilização de NF-e é um processo constituído por diversas etapas que normalmente leva um tempo considerável para ser concluído uma vez que a NF-e envolve diversos fatores e ferramentas. Uma vez que as etapas de elaboração, assinatura e validação da NF-e são concluídas, a próxima que deve ser efetuada é a do envio da NF-e que foi acompanhada nesta terceira parte deste mini-curso. Para permitir o envio e autorização da nota, existe a necessidade de efetuar o consumo de um WebService. O WebService, como comentado anteriormente, é um recurso muito importante na atualidade, pois minimiza a distância e o tempo para a transferência e consulta de informações, utilizando a Internet como base, operando de maneira automatizada. A Secretaria da Fazenda disponibiliza diversos WebServices, sendo cada um deles destinado para uma função específica.Como pôde ser observado também neste artigo, um WebService pode ser facilmente consumido através da utilização de um objeto THTTPReqResp e uma mensagem SOAP.

No próximo artigo serão abordados outros WebServices da NF-e, o envio do e-mail do documento XML e a DANFE.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Download do Manual do Contribuinte e dos Schemas
https://www.fazenda.sp.gov.br/nfe/downloads/arquivos_vigentes.asp

Download de várias cadeias de certificados públicos (CertiSign)
http://www.certisign.com.br/suporte/utilitarios-criptograficos/hierarquias-de-certificacao/icp-brasil

URLs dos Webservices de vários estados
http://www.nfe.fazenda.gov.br/portal/WebServices.aspx

URLs dos WebServices da SEFAZ de SP
https://www.fazenda.sp.gov.br/nfe/url_webservices/url_webservices.asp

DevSpace do Autor
http://www.devmedia.com.br/space.asp?id=283274

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Seu **conhecimento vale muito** na DevMedia

Publique artigos e vídeos gratuitamente no site e participe do programa **PayContent.**

Sua participação pode ser premiada: quanto mais você postar, mais chances de ganhar.

Para saber mais acesse:
www.devmedia.com.br/paycontent

DevMedia

Técnicas Avançadas d

Best Practices, Reflexão, Refator

Assim como qualquer código, um código orientado a objetos mal escrito é tão ruim quanto um código procedural também mal escrito, talvez até pior. A orientação a objetos vai muito além de simplesmente criar uma classe, declarar suas propriedades, alguns métodos e então utilizar um objeto por todo o sistema. É preciso organizar como esses objetos se comunicam, ter o cuidado para que uma classe não faça mais do que ela precisa fazer, e se as camadas lógicas (camada de apresentação, lógica e acesso a dados) não estão se misturando de forma errada, que crie uma dependência entre elas. Existem técnicas, boa práticas e estudos que podem ser aplicados para garantir a boa qualidade de um sistema que faz uso da orientação a objetos. Podemos citar padrões de projeto, uso de interface, abstrações, métodos anônimos e muito mais. Vamos abordar essas boas práticas com exemplos que facilitam o entendimento.

Resumo DevMan

De que trata o artigo:

O artigo trata de apresentar alguns estudos de caso e em cima deles demonstrar a aplicação de boas práticas de Programação Orientada a Objetos com Delphi, incluindo o uso de Fábricas de Construtores, Design Patterns e Interfaces.

Em que situação o tema é útil:

O uso de Boas Práticas, como a aplicação de Padrões de Projeto, ajuda a criar softwares mais fáceis de manter, modificar e evoluir, reduzindo custos e esforço. A aplicação de técnicas avançadas de POO é útil para tirar o máximo de proveito dos recursos da linguagem ao criar sistemas mais flexíveis e modulares, com baixo acoplamento.

Técnicas Avançadas de POO com Delphi:

A primeira parte deste artigo vai mostrar como criar objetos de forma correta com o Delphi, usando por exemplo Métodos Estáticos de Fábrica, Singletons e Fábricas, usando tipos levemente acoplados. A seguir, vamos examinar um recurso do Delphi presente desde suas primeiras versões, que também ajuda a criar softwares mais flexíveis e reduz código, facilitando a manutenção, é a técnica de reflexão. Outro tema intimamente relacionado com Boas Práticas, também abordado neste artigo, é a refatoração, que permite modificar a estrutura (forma) interna do aplicativo, seja para melhorar sua legibilidade ou design, mantendo seu comportamento (behavior) externo. Técnicas avançadas de POO também são aplicadas na prática nos estudos de caso deste artigo, como o uso de Interfaces, Abstração, Encapsulamento, Polimorfismo, Delegates e Métodos Anônimos.

Métodos Estáticos de Fábrica

Uma boa prática é nunca utilizar diretamente os operadores new (Prism) ou Create (Win32) para criar objetos a partir de suas classes concretas. Isso pode causar um forte acoplamento entre elas, o que diminui o reaproveitamento de código e causa dependência que mais tarde pode ocasionar problemas. Uma saída é considerar o uso de fábricas ou métodos estáticos de fábrica. Um exemplo está mostrado na **Listagem 1**.

A aplicação de Console consome uma classe PessoaFisica através de uma fábrica, não possuindo um relacionamento direto de dependência. A fábrica possui um método estático através do uso da palavra reservada class do Delphi, logo ela não precisa ser instanciada. O Create está encapsulado dentro da fábrica, não sendo exposto. Este exemplo pode ser melhorado, aplicando-se uma interface para reduzir ainda mais o acoplamento (isso é feito em outro exemplo adiante).

Singletons

Uma boa técnica seria esconder a instância do objeto dentro da própria classe, juntamente com o construtor, expondo apenas um método público que devolve esta instância interna. Esta técnica é muitas vezes chamada também de Singleton, que possui várias formas de apresentação, dependendo da linguagem e recursos utilizados. Um exemplo está na **Listagem 2**.

Nota do DevMan

A palavra reservada "strict" foi introduzida no Delphi 7 Preview Compiler para .NET e está presente no Win32. Ela termina com a amizade entre classes na mesma unit, uma exclusividade do Pascal, onde dadas uma Classe A e B residentes na mesma Unit, podem enxergar seus atributos e métodos privados, violando o encapsulamento.

Fábricas

Uma outra técnica bem interessante é criar uma fábrica que devolve múltiplos tipos de objetos de dadas classes concretas que possuem um ancestral comum, de forma que possam se aplicar técnicas como herança, polimorfismo e interfaces.

e POO com Delphi
ação e Design Patterns

GUINTHER PAULI

Listagem 1. Método estático de fábrica

```
program Exemplo1;

{$APPTYPE CONSOLE}

{$R *.res}

uses
  System.SysUtils;

type
  // Classe PessoaFisica com atributos simples
  PessoaFisica = class
  strict private
    FNome: string;
    FCPF: string;
  public
    property Nome: string read FNome write FNome;
    property CPF: string read FCPF write FCPF;
  end;

  // Fábrica que constrói pessoas físicas
  Fabrica = class
  public
    class function CriaPessoa: PessoaFisica;
  end;

{ Fábrica }

class function Fabrica.CriaPessoa: PessoaFisica;
begin
  result := PessoaFisica.Create();
end;

var
  GP: PessoaFisica;
begin
  // Uso da fábrica ao invés de chamar create diretamente
  GP := Fabrica.CriaPessoa();
  GP.Nome := 'Guinther Pauli';
  GP.CPF := '123456789-00';
  Write(GP.Nome,' - ',GP.CPF);
end.
```

Listagem 2. Escondendo o construtor e devolvendo uma instância estática

```
program Exemplo2;

{$APPTYPE CONSOLE}

{$R *.res}

uses
  System.SysUtils;

type
  // Classe PessoaSocial com atributos simples
  PessoaSocial = class
  strict private
    FNome: string;
    FBlog: string;
    FFacebook: string;
    FTwitter: string;
    // escondendo o construtor
    constructor Create();
    class var
      _INSTANCE: PessoaSocial;
  public
    property Nome: string read FNome write FNome;
    property Blog: string read FBlog write FBlog;
    property Facebook: string read FFacebook write FFacebook;
    property Twitter: string read FTwitter write FTwitter;
    // Método estático
    class function GetInstance: PessoaSocial;
  end;

var
  GP: PessoaSocial;

{ PessoaSocial }

constructor PessoaSocial.Create;
begin
  // sem implementação
end;

class function PessoaSocial.GetInstance: PessoaSocial;
begin
  if _INSTANCE = nil then
    _INSTANCE := PessoaSocial.Create();
  result := _INSTANCE;
end;

var
  GP : PessoaSocial;
begin
  // Chamando GetInstance ao invés de Create diretamente
  GP := PessoaSocial.GetInstance();
  GP.Nome := 'Guinther Pauli';
  GP.Blog := 'http://gpauli.com';
  GP.Facebook := '/guintherpauli';
  GP.Twitter := '@guintherpauli';
  write(GP.Nome,' - ',GP.Blog,' - ',GP.Facebook,GP.Twitter);
end.
```

A ideia consiste em solicitar a uma fábrica um tipo genérico passando um indicador de baixo acoplamento para ela ("leve"), por exemplo uma flag ou um integer, ou ainda um enum, que não vão causar problemas de dependência. Internamente, a fábrica faz um teste no que foi solicitado e instancia a classe apropriada, devolvendo um objeto de um tipo concreto em uma referência de um tipo mais abstrato.

Vamos examinar o exemplo da **Listagem 3**, que representa um pequeno framework para modelar objetos de Meio de Transporte. A classe base abstrata MeioTransporte define o comportamento comum a todos os tipos específicos, que através de herança são definidos como Carro, Aviao e Bus. Um método abstrato polimórfico é definido na classe base (Ligar), sendo sobrescrito (override) nas classes descendentes. Como o objetivo aqui é o estudo de boas práticas, arquitetura e design, não se faz necessário implementar os métodos, apenas declarar. Aqui apenas devolvemos uma string para fins de debug.

Nota do DevMan

Uma boa prática para não se incluir código de debug e instrumentação misturado ao código lógico a aplicação é usar uma ferramenta como o CodeSite, agora integrado ao IDE do Delphi, como mostrou o Rodrigo Araujo na ClubeDelphi edição 125. CodeSite é a melhor ferramenta de Logging e Debug para Delphi. Mas o Delphi já não tem um excelente depurador integrado? Para que eu preciso de outro depurador? O CodeSite pode ser classificado como um eficaz sistema de Logging, ou ainda, de instrumentação de código, também podendo atuar na técnica de Profiling. Imagine que estamos desenvolvendo uma grande aplicação, com milhares de linhas de código, classes de negócio, componentes, acesso a dados, pacotes, interfaces VCL etc. É muito comum nos remetermos ao que chamamos de "depurador"dos pobres quando é preciso inspecionar o próprio comportamento da aplicação em execução (o que foi feito no exemplo aqui). Por exemplo, o envio de informações para a tela ou para um arquivo, sobre consumo de memória, ou tempo de execução de procedimentos, a fim de detectar gargalos, verificar estados de objetos, alcance de pontos de execução do sistema e mais. Isso se torna ainda mais claro em ambientes de produção com servidores DataSnap ou Web, tipicamente aplicações multithread, onde simplesmente não temos como depurar uma aplicação em uma situação real de execução, pois: 1) não é simples depurar um servidor rodando remotamente com o IDE na máquina de desenvolvimento, a saída seria o Remote Debugger 2) é praticamente impossível debugar aplicações multithread, ainda mais se usarem eventos. O CodeSite permite que o desenvolvedor "injete" código de inspeção (vamos chamar de um ShowMessage "tunado"), onde é possível enviar mensagens, objetos e tipos personalizados para um ambiente real de log ("ao vivo"), podendo ainda ser analisado posteriormente.

A **Listagem 4** define a fábrica. Ela possui um único método estático chamado CreateTransporte, que recebe como parâmetro uma "flag", uma referência a um tipo enumerado. Essa técnica vai ajudar a classes externa, que precisarem de meio de transportes concretos, a passar um tipo "leve" primitivo, sem causar forte dependência (o que seria um "bad smell"). Conforme o tipo passado, a fábrica efetua o teste com case e criar internamente as classes concretas, não expondo qualquer informação de criação ao meio externo.

"Bad Smell" é um termo muito utilizado em design e arquitetura de software para indicar um código que está "cheirando mal". Nesse sentido, criar um bom design de código é prepará-lo para mudanças. Aplicações mal escritas são muito rígidas, difíceis de sofrerem modificações, mesmo com a ajuda de técnicas de refatoração (veremos adiante). Você pode levar em conta as seguintes considerações para avaliar se sua aplicação Delphi está "fedendo":

Listagem 3. Pequeno framework para modelar Meios de Transporte

```
...
  // Classe Genérica Abstrata
  MeioTransporte = class abstract
  public
   function Ligar(): string; virtual; abstract;
  end;

  // Classe Concreta derivada
  Carro = class(MeioTransporte)
   function Ligar(): string; override;
  end;

  // Classe Concreta derivada
  Aviao = class(MeioTransporte)
   function Ligar(): string; override;
  end;

  // Classe Concreta derivada
  Bus = class(MeioTransporte)
   function Ligar(): string; override;
  end;

implementation

{ Carro }

function Carro.Ligar(): string;
begin
 result := 'Método Ligar de Carro chamado';
end;

{ Aviao }

function Aviao.Ligar(): string;
begin
 result := 'Método Ligar de Aviao chamado';
end;

{ Bus }

function Bus.Ligar(): string;
begin
 result := 'Método Ligar de Bus chamado';
end;
```

Listagem 4. Uma fábrica mais sofisticada

```
...
type
 // Enum simples para reduzir o acoplamento
 TipoMeioTransporte = (mtCarro, mtAviao, mtBus);

 // Fábrica
 MeioTransporteFactory = class
 public
  // método estático de fábrica
  class function
  CreateTransporte (Tipo: TipoMeioTransporte): MeioTransporte;
 end;
{ MeioTransporteFactory }
class function MeioTransporteFactory.CreateTransporte(
 Tipo: TipoMeioTransporte): MeioTransporte;
begin
 case Tipo of
  mtCarro: result := Carro.Create();
  mtAviao: result := Aviao.Create;
  mtBus: result := Bus.Create;
 end;
end;
```

• Rigidez: a alteração de um simples bloco de código tem um impacto devastador em toda as units da aplicação;
• Fragilidade: uma alteração em um código faz a aplicação apresentar defeitos em outros locais;
• Complexidade: é quase impossível modificar / evoluir o software, seja para uma nova versão do Delphi, mudança de arquitetura de camadas, pois ninguém entende o que código faz ou deveria fazer;
• Duplicação: código redundante em várias units, dificultando a refatoração ao invés de ajudá-la.

Quase todas as técnicas para ser criar um bom código ("cheiroso") se resumem na criação de estruturas com baixo acoplamento, seja aplicando Design Patterns, Refatoração, Reflexão e fazendo-se uso intenso de interfaces, todos temas deste artigo.

Reflexão

Reflection, ou reflexão, é um poderoso recurso presente na maioria das linguagens orientadas a objetos que permite inspecionar metadados de objetos em tempo de execução, o que inclui, por exemplo, informações sobre tipos, classes, interfaces e enumerações. O recurso é amplamente utilizado em várias situações. Por exemplo, aplicações que permitem que o usuário altere alguma configuração em um controle de tela, personalizando conforme sua necessidade, podem usar Reflection para isso. Frameworks de persistência normalmente recebem objetos da camada de apresentação, necessitando inspecionar este objeto buscando em runtime informações sobre nomes de propriedades, para então gerar comandos SQL dinamicamente. Isso pode ser feito com Reflection. Muitas vezes o uso do recurso vai de encontro à utilização de Generics, onde temos objetos fortemente tipados. Outros preferem ligar objetos em runtime através de interfaces, uma boa prática. Acredito que a obtenção de uma boa arquitetura de software, expansível, plugável e flexível, seja a combinação harmoniosa de Reflection, Generics e Interfaces.

No Delphi, o mecanismo de reflexão é também conhecido como RTTI – Runtime Type Information. A **Listagem 5** mostra como podemos usar esta técnica para obter os nomes em formato string dos tipos declarados na enumeração, reduzindo codificação. O resultado é que os nomes dos tipos de meio de transporte vão aparecer automaticamente no combobox. Se um novo valor for incluído no enum do código, ele aparece automaticamente no combo, pois a informação é obtida em tempo de execução. Note que aqui estamos usando a RTTI "old-school", baseada na unit TypInfo, mas vale lembrar que a partir do Delphi 2010 a RTTI foi remodelada e melhorada.

A ideia chave neste exemplo é que o usuário possa escolher o meio de transporte desejado no combobox, que é preenchido dinamicamente. A seguir, ao clicar em um botão, o método polimórfico "Ligar" é chamado a partir da classe abstrata, e através de polimorfismo, seja identificado em runtime qual o método de mesmo nome de uma classe derivada deve ser realmente invocado, porém, sem causar uma forte dependência (como se fôssemos utilizar por exemplo RTTI com vários if's, "is" e "as", um para cada tipo). O formulário da aplicação é apresentado adiante, na **Figura 1.**

Fabricando objetos e chamando métodos polimórficos

No exemplo, o formulário possui uma referência para um meio de transporte, através de uma propriedade, como mostra a **Listagem 6**. Note que isso causa um menor acoplamento entre a classe do formulário principal e as classes "alvo", ou seja, as concretas, que realmente possuem a lógica do negócio. Estamos nos comunicando com um nível de abstração maior, ou seja, através da classe abstrata de nível superior, que não tem implementação, apenas um método virtual.

Listagem 5. Reflexão - Preenchendo ComboBox com dados do Enum

```
procedure TFrmMain.FormCreate(Sender: TObject);
var
 i: Integer;
begin
 // Usa reflexão para obter nomes em string do tipo enumerado
 // uses TypInfo
 for i := ord(low(TipoMeioTransporte))
       to ord(high(TipoMeioTransporte)) do
          cbTipo.Items.Add(GetEnumName(TypeInfo(TipoMeioTransporte),i));
end;
```

Listagem 6. Reflexão - Preenchendo ComboBox com dados do Enum

```
...
TFrmMain = class(TForm)
...
private
 FMeioTransporte: MeioTransporte;
public
...
property Transporte: IMeioTransporte
          read FMeioTransporte write FMeioTransporte;
...
```

Agora vamos examinar bem o código da **Listagem 7.** Primeiro, nós obtemos o valor do tipo enumerado a partir de um integer, fazendo um cast para ver o que o usuário selecionou no combobox. A seguir, solicitamos à fábrica que devolva um tipo concreto baseado nessa "flag". O método Ligar, polimórfico e comum a todos os meios de transporte é então invocado. Note que neste momento, o compilador do Delphi não sabe o que está dentro da referência Transporte, pois está declarado como um tipo abstrato no formulário (MeioTransporte). É o polimorfismo que garante que o método correto seja chamado em runtime. Basicamente, ele está declarado como abstract apenas para poder ser chamado pelo compilador e reduzir o acoplamento no framework, caso contrário, para cada novo meio de transporte que fosse criado (lembre-se, mudanças ocorrem), mais um teste precisaria ser feito e novo código injetado. Ao final, a mensagem de debug é colocada em um memo para fins de profiling.

O diagrama UML da **Figura 2**, gerada pela ferramenta Together integrada ao Delphi desde a versão 2005, permite visualizar o baixo acoplamento entre as classes da aplicação, incluindo a fábrica, formulário principal, tipo enumerado, classe abstrata pai e classes concretas descendentes.

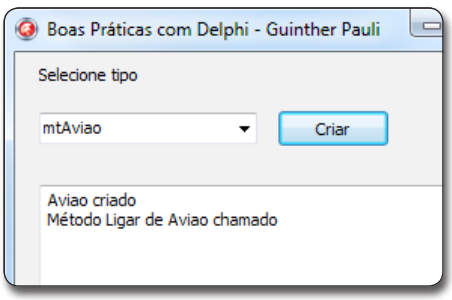


Figura 1. Formulário principal permite escolher um tipo leve e invocar um método polimórfico

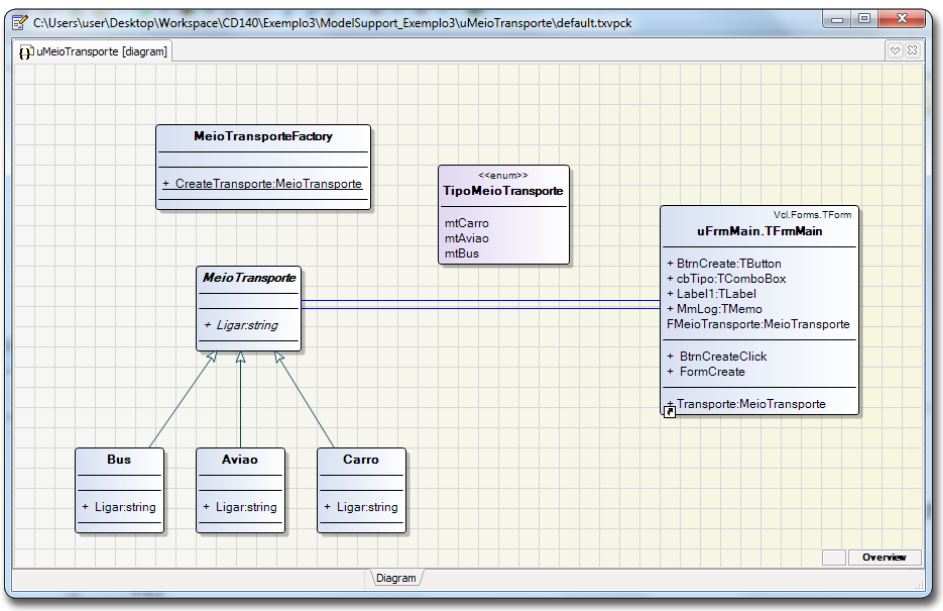


Figura 2. Diagrama UML das classes do projeto

Refatorando com Extract Interface

Podemos descrever a Refatoração como a transformação de uma representação para outra mantendo o mesmo nível da abstração, preservando o comportamento externo (behavior). Uma transformação na estrutura pode ser frequentemente relacionado à sua aparência (por exemplo melhorar a legibilidade), mas pode ser uma alteração no design da aplicação. Refatorações podem melhorar a qualidade de um software, por exemplo, extensibilidade, modularidade, maior reutilização de código, menor complexidade, manutenção mais fácil, eficiência etc. Podemos dividir as refatorações em duas grandes categorias, as primitivas e as formados por composição. Uma refatoração primitiva seria por exemplo renomear um objeto, mover um método etc. As refatorações mais complexas são normalmente obtidas pela composição de várias primitivas.

O processo de refatoração consiste em uma série de atividades distintas, incluindo: identificar *onde* um software precisa ser refatorado, determinar *qual* refatoração pode ser aplicada, *garantir* que o comportamento se mantém, aplicar o refactoring, *medir* o impacto do refactoring nas características do software, como complexidade, usabilidade, custo, esforço etc. *Manter a consistência* entre o código refatorado e outros artefatos do sistema, como diagramas UML, documentação em HTML, PDF, especificação de requisitos, testes etc.

Identificar "Bad Smells" no código para criar oportunidades de refatoração é uma técnica bastante interessante e difundida atualmente. Podemos por exemplo usar métricas de software para, por exemplo, identificar falhas no design de um software que apresenta classes de um framework com um alto grau de acoplamento, reduzindo assim o reaproveitamento. Neste caso, poderia se usar refatoração para padrões de projeto, a fim de melhorar o design. Testes ou gráficos de dependência podem ajudar a medir a eficiência dos refactorings. Outro ponto importante a ser considerado são as pré e pós condições das atividades de refatoração. Por exemplo, uma pré-condição de um sistema de tempo real, como uma aplicação Web de E-Commerce, é o tempo de resposta ao usuário. De nada adianta melhorar a legibilidade do código e design se ao final tivermos uma queda no tempo de resposta, pois nesse caso, *performance é uma pré-condição*. O mesmo vale para sistemas embarcados, por exemplo, aplicações que rodam em Smartphones, elas precisam consumir pouca memória e bateria, outra pré-condição.

Em um framework de classes com características, uma boa oportunidade de refatoração seria criar um "denominador comum" para elas, para que classes externas pudessem consumir recursos da hierarquia com um nível maior de abstração, nesse caso, uma interface seria o ideal. Existe uma refatoração para isso, chamada Extract

Listagem 7. Solicitando objetos concretos da fábrica passando um tipo leve e chamando métodos polimórficos

```
procedure TFrmMain.BtrnCreateClick(Sender: TObject);
var
  // Referência "thin"
  Tipo: TipoMeioTransporte;
  str: string;
begin
  // "Cast" de ComboBox para Enum
  Tipo := TipoMeioTransporte(cbTipo.ItemIndex);
  // Solicita tipo concreto à fábrica
  Transporte := MeioTransporteFactory.CreateTransporte(Tipo);
  // Chamada método polimórfico da classe abstrata
  str := Transporte.Ligar();
  MmLog.Lines.Add(Transporte.ToString() + ' criado');
  MmLog.Lines.Add(str);
end;
```

Interface. Ela permite selecionar membros de uma dada classe e a partir dela extrair o esqueleto de uma interface, ao mesmo tempo que já adiciona os itens necessários para sua implementação.

No Delphi, o suporte a refatoração existe desde o Delphi 2005. Em nosso estudo de caso, podemos implementar a refatoração Extract Interface a fim de reduzir ainda mais acoplamento existente entre o formulário e as classes concretas finais, criando um mecanismo leve de contrato entre ambas as partes. A **Figura 3** mostra a opção utilizada, observando que deve se manter selecionado no editor o nome da classe alvo desejada. A **Figura 4** mostra a interface que será gerada e a **Figura 5** um preview do que será afetado, onde você deve confirmar a refatoração. Ao final, seu código deve estar como mostram as **Listagens 8, 9** e **10**. Repare que aqui estamos mantendo dois níveis de abstração, aqueles propiciados pela interface, e o outro proporcionado pela classe abstrata. Normalmente os dois são combinados, como fizemos aqui, pois a classe abstrata pode implementar um código comum a todas as subclasses, enquanto que a interface não pode conter qualquer tipo de implementação, apenas especificar o contrato. Note que a factory agora faz uso da interface ao invés da abstração.

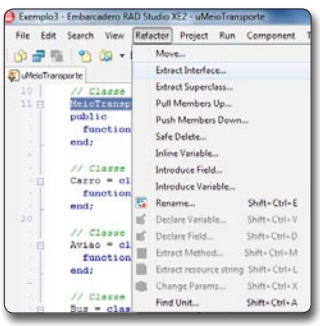


Figura 3. Opção para Refatoração

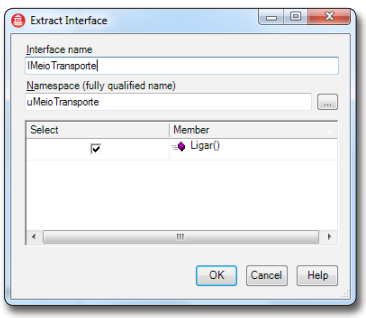


Figura 4. Extract Interface

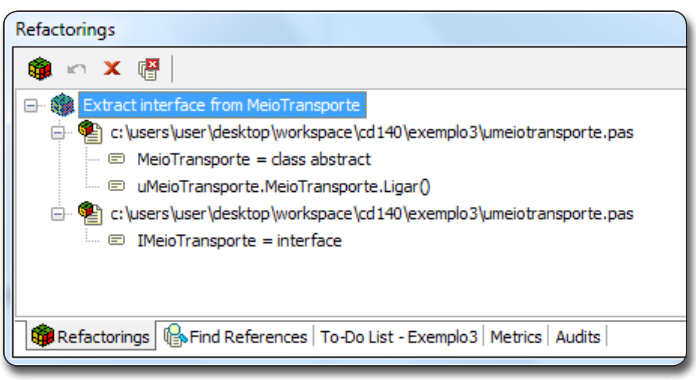


Figura 5. Extract Interface

Mais Boas Práticas e um novo Estudo de Caso

Para aplicar mais algumas Best Practices, vamos considera agora um outro estudo de caso. A ideia deste exemplo é modelar um sistema de notificação de clientes de uma empresa, por exemplo, uma operadora de telefonia ou um banco. Tais notificações podem ser enviadas por SMS, Email e VoiceMail. Um requisito da aplicação é que ela esteja preparada para futuras formas de notificação que a empresa possa vir a adotar, ou seja, precisamos criar o design de um software bem preparado para futuras mudanças.

A modelagem inicial do framework contém "bad smells", como pode ser visto na **Listagem 11.** Note que a classe de Notificacao possui uma dependência rígida e direta com a classe Email, podendo enviar mensagens ao cliente somente através deste meio. Porém existe um design um tanto elegante, escondemos a forma de envio encapsulando a referência que está interna na classe Notificacao, na sessão private, favorecendo um bom princípio da OO, o encapsulamento.

A **Listagem 12** mostra uma aplicação de Console que consome a classe de Notificacao. Um aviso é enviado ao cliente informado a data de vencimento da fatura. A saída final da aplicação é algo do tipo:

```
Email enviado: Sua fatura vence dia 29/04
```

Listagem 8. Interface gerada

```
IMeioTransporte = interface
  function Ligar(): string;
end;
```

Listagem 9. Classe abstrata herda de TInterfacedObject e implementa IMeioTransporte

```
MeioTransporte = class abstract(TInterfacedObject, IMeioTransporte)
public
  function Ligar(): string; virtual; abstract;
end;
```

Listagem 10. Fábrica agora usa interface

```
// Fábrica
MeioTransporteFactory = class
public
  // método de fábrica
  class function CreateTransporte
        (Tipo: TipoMeioTransporte) : IMeioTransporte; Repate a Interface
end;
```

Listagem 11. Framework de notificação, primeira versão

```
unit uNotificao; interface
uses
  uFormasNotificacao;
type
  Email = class
    procedure Avisar(Texto: string);
  end;
  Notificacao = class
  private
    FEMail: Email;
  public
    constructor Create;
    procedure Send(Texto: string);
  end;
implementation
{ Notificacao }
procedure Notificacao.Send(Texto: String);
begin
  FEmail.Avisar(Texto);
end;
constructor Notificacao.Create;
begin
  inherited; FEMail := Email.Create();
end;
{ Email }
procedure Email.Avisar(Texto: string);
begin
  Writeln('Email enviado: ' + Texto);
end;
end.
```

Listagem 12. Aplicação consumindo classe de Notificacao – Aplicação de Console

```
program Exemplo4;

{$APPTYPE CONSOLE}

{$R *.res}

uses
  System.SysUtils,
  uNotificao in 'uNotificao.pas';

var
  Aviso: Notificacao;
begin
  Aviso := Notificacao.Create();
  Aviso.Send('Sua fatura vence dia 29/04');
end.
```

Vamos estender nosso framework de notificação, a fim de proporcionar novas formas de envio. A **Listagem 13** mostra como ficou a nova versão. Declaramos uma interface base para todos os tipos, que agora incluem SMS e VoiceMail, além do Email, todos colocados em uma unit a parte. As classes concretas implementam a interface base, herdando de TInterfacedObject, uma classe do Delphi que já implementa os métodos básicos de IInterface, como as responsáveis pela contagem de referência de objetos, livrando o programador desta tarefa. A **Listagem 14** mostra como ficou a classe de Notificacao, ainda dependente de Email. Substituímos a aplicação de Console por um formulário (**Figura 6, Listagem 15**), onde o usuário pode informar a notificação e clicar no botão para enviá-la ao cliente. A **Figura 7** mostra a mensagem enviada após o botão ser clicado. Nosso código está cheirando mal, precisamos suportar novas formas de envio no form, permitindo que essa escolha seja feita pelo usuário em runtime, e ainda resolver um problema sério de dependência.

Listagem 13. Framework de notificação, segunda versão

```
unit uFormasNotificacao;
interface
uses
  Dialogs;
type
  IFormaNotificacao = interface
    procedure Avisar(Texto: String);
  end;

  Email = class(TInterfacedObject,IFormaNotificacao)
  public
    procedure Avisar(Texto: String);
  end;

  SMS = class(TInterfacedObject,IFormaNotificacao)
  public
    procedure Avisar(Texto: String);
  end;

  VoiceMail = class(TInterfacedObject,IFormaNotificacao)
  public
    procedure Avisar(Texto: String);
  end;

implementation

{ Email }

procedure Email.Avisar(Texto: String);
begin
  ShowMessage('Email enviado: ' + Texto);
end;

{ SMS }

procedure SMS.Avisar(Texto: String);
begin
  ShowMessage('SMS enviado: ' + Texto);
end;

{ VoiceMail }

procedure VoiceMail.Avisar(Texto: String);
begin
  ShowMessage('Voice mail gravado: ' + Texto);
end;

end.
```

Listagem 14. Classe de Notificacao, nova versão, ainda dependente de classe concreta

```
unit uNotificacao;

interface

uses
  uFormasNotificacao;

type

  Notificacao = class
  private
    FEMail: Email;
  public
    constructor Create;
    procedure Send(Texto: string);
  end;

implementation

{ Notificacao }

procedure Notificacao.Send(Texto: String);
begin
  FEmail.Avisar(Texto);
end;

constructor Notificacao.Create();
begin
  inherited;
  FEMail := Email.Create();
end;

end.
```

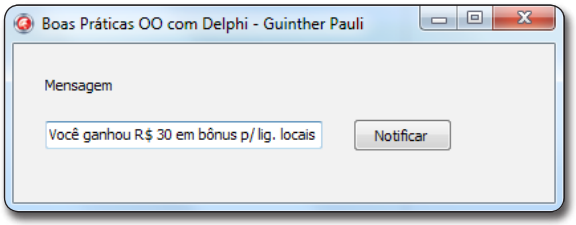


Figura 6. Formulário de notificação

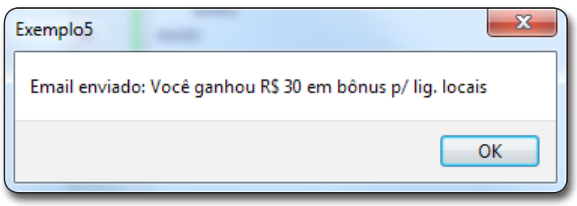


Figura 7. Mensagem de aviso enviado com sucesso

Injeção de Dependência (DI) / Inversão de Control (IoC)

Conceber aplicações de software bem estruturadas e flexíveis a mudanças tem sido um dos grandes desafios impostos à área de desenvolvimento de sistemas no decorrer dos anos. As organizações estão cada vez mais sujeitas a transformações repentinas e de profundo impacto. Isto acaba por exigir dos profissionais de Tecnologia da Informação uma resposta rápida ao atendimento de novas demandas ou, ainda, à adaptação de soluções pré-existentes.

Prazos exíguos, equipes reduzidas, falta de planejamento, inexistência de uma modelagem para a solução a ser entregue, dentre outros aspectos, quase sempre resultarão em codificação de baixa qualidade para a solução que será obtida. Isto pode se constituir, futuramente, num verdadeiro pesadelo em situações que exijam adaptações de funcionalidades já criadas anteriormente.

A utilização da técnica de desenvolvimento em camadas, prática amplamente difundida no mercado, procura fornecer meios para a criação de aplicações mais flexíveis. Entretanto, a simples utilização deste tipo de abordagem não garante por si só resultados positivos. Ainda existirá a possibilidade de se chegar a um produto que faz uso de classes de objetos com um alto grau de acoplamento; com isto, ajustes que na teoria seriam simples levam a mudanças drásticas em todo um software, aumentando ainda a chance de se introduzirem erros inesperados na solução. A Injeção de Dependência auxilia a resolver esse dilema, desacoplando as camadas criadas.

Primeiramente, deve-se levar em conta que a Injeção de Dependência (DI, do inglês Dependency Injection) é na verdade, uma aplicação específica de um conceito mais abrangente conhecido como Inversão de Controle (IoC, do inglês Inversion of Control).

A Inversão de Controle é uma técnica na qual um objeto delega a execução de uma atividade a outra classe, ao invés de tentar controlar diversos aspectos que fugiriam do objetivo inicial para o qual havia sido definido. Esta característica é particularmente importante, já que permite o desenvolvimento de classes com um maior grau de coesão, visto que não acumulam um grande número de responsabilidades, o que do contrário dificultaria a manutenção futura das mesmas e resultaria em um software mais propenso a falhas.

Listagem 15. Formulário usa a classe Notificacao para despachar mensagens

```
unit uFrmMain;

interface

uses
 Winapi.Windows, Winapi.Messages,
      System.SysUtils, System.Variants, System.Classes,
    Vcl.Graphics, Vcl.Controls, Vcl.Forms,
 Vcl.Dialogs, Vcl.StdCtrls, uNotificacao;

type
 TFrMain = class(TForm)
  edtTexto: TEdit;
  Label1: TLabel;
  BtnNotificar: TButton;
  procedure BtnNotificarClick(Sender: TObject);
 private
  FNotificacao: Notificacao;
  { Private declarations }
 public
   { Public declarations }
   property Aviso: Notificacao read FNotificacao write FNotificacao;
 end;

var
 FrMain: TFrMain;

implementation

{$R *.dfm}

 procedure TFrMain.BtnNotificarClick(Sender: TObject);
 begin
  Aviso := Notificacao.Create();
  try
   Aviso.Send(edtTexto.Text);
  finally
   Aviso.Free;
  end;
 end;

end.
```

O uso de injeção de dependência também é uma forma de se aplicar o conceito de inversão de controle. Neste caso ao invés de um objeto instanciar todas as classes das quais depende, o mesmo delega a outro mecanismo esta tarefa (container). Ao se optar pela utilização de um container o mesmo será responsável pelas seguintes atividades: Analisar de quais tipos de objetos uma classe depende; Criar os objetos a serem consumidos; Associar as instâncias de objetos criadas à classe consumidora, sendo que a isto se chama "resolver" as dependências.

Dentre as vantagens da adoção de um framework de injeção de dependência, merece ser destacado: Foco no desenvolvimento voltado ao uso de interfaces, com a obtenção de aplicações em que os componentes tendem a ser mais coesos, além de estarem fracamente acoplados entre si, já que o conceito de Injeção de

Dependência privilegia o desenvolvimento de software em camadas; Flexibilidade diante da necessidade de mudanças, devido à redução na dependência entre as partes que constituem uma aplicação; O desenvolvimento de software seguindo boas práticas, uma vez que os conceitos de Inversão de Controle e Injeção de Dependência promovem o uso de patterns amplamente difundidos e utilizados com sucesso pela comunidade de desenvolvedores; Evita-se a duplicação de código instanciando classes e que estaria espalhado por toda uma solução.

Em nosso estudo de caso, como já temos um framework modelado para suportar *n* formas de envio de notificação, vamos adaptar a classe Notificacao para que não faça referência mais a uma classe concreta de Email, mas sim a uma estrutura mais flexível e plugável, a interface (**Listagem 16**). Além disso, o objeto concreto que enviará a notificação não será mais criado internamente pela classe de Notificacao, ele virá prontinho de fora, da própria classe do formulário (ou seja, a preferência é do cliente). Acabamos de inverter o controle da situação (IoC). A referência é recebida no construtor e guardada internamente, para que possa ser chamada em momento oportuno, mais precisamente, quando o método Send de Notificacao for invocado chamamos então o método de interface Avisar que resolverá qual classe concreta será usada (como o polimorfismo).

Listagem 16. Notificacao agora recebe a forma no construtor, externamente (a dependência foi invertida)

```
unit uNotificacao;

interface

uses
  uFormasNotificacao;

type

  Notificacao = class
  private
    FFormaNotificao: IFormaNotificacao;
  public
    constructor Create(AFormaNotificacao: IFormaNotificacao);
    procedure Send(Texto: string);
  end;

implementation

{ Notificacao }

procedure Notificacao.Send(Texto: String);
begin
  FFormaNotificao.Avisar(Texto);
end;

constructor Notificacao.Create(AFormaNotificacao: IFormaNotificacao);
begin
  inherited Create();
  // repassa instânncia do create para referência interna
  FFormaNotificao := AFormaNotificacao;
end;

end.
```

Figura 8. Formulário agora permite escolha da forma de envio da mensagem

A **Figura 8** mostra o formulário principal. A **Listagem 17** trata de identificar qual a forma de envio foi escolhida pelo usuário, cria então a classe concreta e repassa essa como parâmetro do Create de Notificacao, que recebe qualquer possível tipo de IFormaNotificacao, hoje SMS, Email e VoiceMail, mas prepara para mudanças. Note que aqui poderíamos ter aplicado uma fábrica no bloco que cria os objetos, como feito em estudo de caso anterior. Além disso, poderíamos usar um enum ou a nova RTTI para obter automaticamente o nome das classes que implementam IFormaNotificacao, preenchendo o Radio. A **Figura 9** mostra como ficou a dependência entre as classes.

Bom, a partir daqui pediria que o amigo leitor esquece-se um pouco das técnicas que aplicamos até aqui, até porque muitos condenam a prática que irei apresentar a seguir. Meu objetivo aqui é abrir uma discussão / paralelo entre Delegates vs. Polimorfismo, e também entre Métodos Anônimos vs. Abstrações. E também alavancar o uso de delegates e métodos anônimos em Delphi, visto que na plataforma .NET, suportado pelo Prism, o recurso vem sendo amplamente utilizado, principalmente com LINQ e Expressões Lamda, para tornar o código mais simples e fluente.

Delegates

Em nosso estudo de caso, precisamos rever alguns conceitos. Por que criamos um framework que comporta *n* formas de notificação? Para suportar futuras formas que venham a surgir sem causar um desagradável efeito dominó em toda aplicação. Existe, porém outra forma de tornar nosso *código flexível* sem recorrer a recursos clássicos da OO, como herança, encapsulamento, polimorfismo, abstração e interfaces. São os delegates.

Apesar de também desenvolvedor C#, vou falar de programador Delphi para programador Delphi, vai facilitar o aprendizado. Quando você dá um duplo clique em um botão e é criado um manipulador para o evento OnClick, chamado algo como Button1Click, temos o seguinte: OnClick é o evento, é um ponteiro para uma implementação, essa implementação é o Button1Click, o seu código. Esse ponteiro tem um tipo, que define a assinatura que seu método deve ter, normalmente com parâmetro Sender: TObject. Esse tipo se chama em outras plataformas, como .NET, de Delegate. No Delphi chamamos de tipo *procedure of object*.

A **Listagem 18** mostra como ficou nosso novo framework de Notificacao. Repare que extinguimos toda a família de classes concretas em prol de um único ponteiro, chamado aqui de FFormaNotificacao, que ao invés de referenciar uma interface que mais tarde apontaria para uma classe concreta que invocaria um método concreto, aponta para o próprio método concreto que contém a lógica da notificação. O Create da classe não recebe mais um tipo concreto compatível com IFormaNotificacao, mas um ponteiro para método, que é guardado internamente para ser chamado quando o usuário for enviar a Notificacao (método Send).

A **Listagem 19** mostra como ficou o código do formulário. Nada de criar classes concretas, elas não existem mais, porém o usuário ainda pode escolher uma forma flexível de enviar notificações, mantendo a inversão de controle. Ao invés disso, temos um novo método na classe do formulário, que inclui a lógica flexível da notificação, que anteriormente era obtida através da aplicação de polimorfismo com interfaces. Ao criar um objeto da classe Notificacao, passamos um ponteiro para esse método, que é então guardado para ser invocado posteriormente. Quando o método Send é chamado, a classe Notificacao invoca de volta o método do formulário. Se você debugar a aplicação dando um Trace Into com um Breakpoint na criação do objeto Aviso, verá bem esse comportamento em ação. É claro, o ideal seria colocar essa lógica fora do formulário, mas meu objetivo aqui foi apenas mostrar o quão simples e prático é usar Delegates no Delphi Win32.

Métodos Anônimos

E finalmente, vou apresentar um recurso que foi introduzido no Delphi 2009, chamado de Método Anônimo (Anonymous Method).

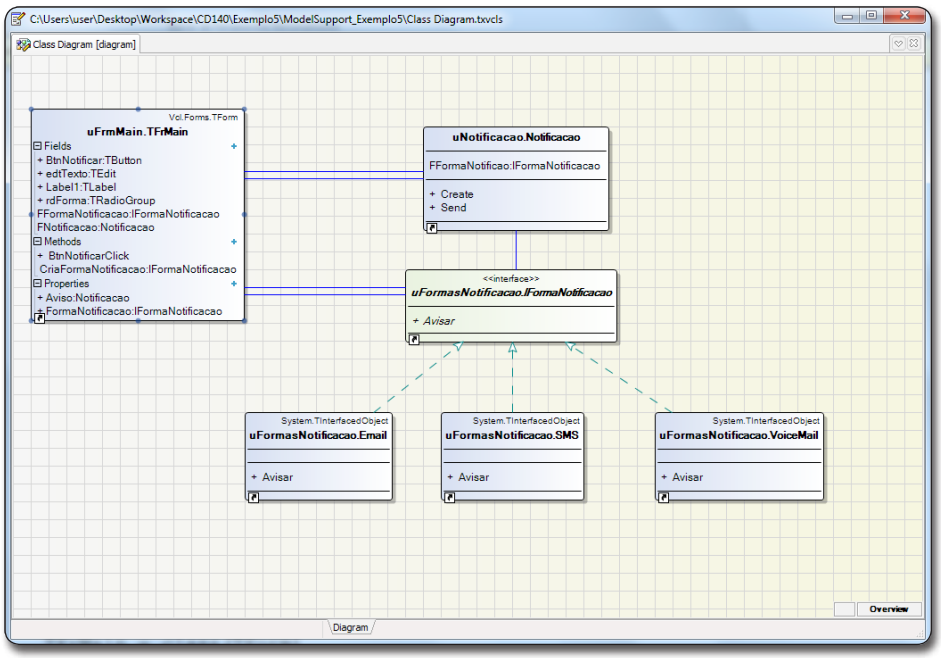


Figura 9. Formulário agora permite escolha da forma de envio da mensagem

Listagem 17. Formulário agora tem a responsabilidade de repassar a forma de envio

```
unit uFrmMain;
interface
uses
  Winapi.Windows, Winapi.Messages, System.SysUtils,
   System.Variants, System.Classes, Vcl.Graphics,
   Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls,
   uNotificacao, Vcl.ExtCtrls, uFormasNotificacao;
type
  TFrMain = class(TForm)
    edtTexto: TEdit;
    Label1: TLabel;
    BtnNotificar: TButton;
    rdForma: TRadioGroup;
    procedure BtnNotificarClick(Sender: TObject);
  private
    FNotificacao: Notificacao;
    FFormaNotificacao: IFormaNotificacao;
    function CriaFormaNotificacao: IFormaNotificacao;
    { Private declarations }
  public
    { Public declarations }
    property Aviso: Notificacao
      read FNotificacao write FNotificacao;
    property FormaNotificacao: IFormaNotificacao
      read FFormaNotificacao write FFormaNotificacao;
end;
var
  FrMain: TFrMain;
implementation
{$R *.dfm}
procedure TFrMain.BtnNotificarClick(Sender: TObject);
begin
  FormaNotificacao := CriaFormaNotificacao();
  Aviso := Notificacao.Create(FormaNotificacao);
  try
    Aviso.Send(edtTexto.Text);
  finally
    Aviso.Free;
  end;
end;
function TFrMain.CriaFormaNotificacao: IFormaNotificacao;
begin
  // poderiaíamos usar uma fábrica aqui
  case rdForma.ItemIndex of
    0: result := Email.Create();
    1: result := SMS.Create();
    2: result := VoiceMail.Create();
  end;
end;
end.
```

Listagem 18. Framework de formas de notificação totalmente eliminado, interface transformada em delegate, que prevalecerá sobre o polimorfismo, sem herança, com preservação do mesmo comportamento (behavior)

```
unit uNotificacao;

interface

uses
  uFormasNotificacao;

type
  // procedure of object em Delphi equivale a um "delegate"
  FormaNotificacaoHandler = procedure (Text: string) of object;

  Notificacao = class
  private
    // isso não é mais uma interface,
    // é um ponteiro para um método
    FFormaNotificao: FormaNotificacaoHandler;
  public
    constructor Create(AFormaNotificacao: FormaNotificacaoHandler);
    procedure Send(Texto: string);
    end;

implementation

{ Notificacao }

procedure Notificacao.Send(Texto: String);
begin
  FFormaNotificao(Texto);
end;

constructor Notificacao.Create(AFormaNotificacao: FormaNotificacaoHandler);
begin
  inherited Create();
  // repassa a referência do método para a variável interna
  FFormaNotificao := AFormaNotificacao;
end;

end.
```

Listagem 19. Injetando código usando delegates ao invés de uma classe concreta

```
unit uFrmMain;
interface
uses
  Winapi.Windows, Winapi.Messages, System.SysUtils,
  System.Variants, System.Classes, Vcl.Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls,
  uNotificacao, Vcl.ExtCtrls, uFormasNotificacao;
type
 TFrMain = class(TForm)
    edtTexto: TEdit;
    Label1: TLabel;
    BtnNotificar: TButton;
    rdForma: TRadioGroup;
    procedure BtnNotificarClick(Sender: TObject);
  private
    { Private declarations }
    FNotificacao: Notificacao;
    // Código a ser injetado no "delegate"
    procedure Avisar(Texto: string);
  public
    { Public declarations }
    property Aviso: Notificacao
      read FNotificacao write FNotificacao;
  end;
var
  FrMain: TFrMain;

implementation
{$R *.dfm}
// A implementação variante, antes obtida por polimorfismo,
// é agora passada por parâmetro usando delegates,
// sendo injetada de fora, sem herança
procedure TFrMain.Avisar(Texto: string);
begin
  ShowMessage(Texto + ' notificado via ' + rdForma.Items[rdForma.ItemIndex]);
end;
procedure TFrMain.BtnNotificarClick(Sender: TObject);
begin
  // Abaixo é passado um ponteiro para o método no construtor
  Aviso := Notificacao.Create(Avisar);
  try
    Aviso.Send(edtTexto.Text);
  finally
    Aviso.Free;
  end;
end;
end.
```

Um M.A. é um "bloco" de código de implementação de um procedimento ou função, porém que não tem assinatura, só a implementação. Pense que você precisa criar um Button em RunTime e precisa manipular o seu OnClick também em runtime, você teria que fazer algo do tipo:

```
Button1.OnClick := Aqui_Vai_O_Nome_Do_Seu_Método;
```

Em um cenário de M.A., poderíamos fazer:

```
Button1.OnClick := procedure (Sender: TOBject)
begin
 // implementação
   end
```

Para usar o recurso no exemplo, é necessário modificar assinatura do "delegate" para:

```
FormaNotificacaoHandler = reference to procedure (Text: string);
```

A **Listagem 20** mostra como ficou a versão final da aplicação, note que não precisamos criar um método na classe do formulário e passar uma referência a ele no construtor. Nós passamos a própria implementação do método como parâmetro do Create. Eu sei que isso vai parecer muito estranho a primeira vista para quem nunca usou o recurso, porém note que não foi necessário parar a codificação para incluir um novo método na classe com um dado nome, implementá-lo e voltar ao Create para referenciá-lo,

Listagem 20. Passando um "pedaço" de código sem nome que contém a lógica de notificação, para ser chamado internamente na classe Notificacao

```
unit uFrmMain;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils,
  System.Variants, System.Classes, Vcl.Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls,
  uNotificacao, Vcl.ExtCtrls, uFormasNotificacao;

type
 TFrMain = class(TForm)
   edtTexto: TEdit;
   Label1: TLabel;
   BtnNotificar: TButton;
   rdForma: TRadioGroup;
   procedure BtnNotificarClick(Sender: TObject);
 private
   { Private declarations }
   FNotificacao: Notificacao;
 public
   { Public declarations }
   property Aviso: Notificacao
     read FNotificacao write FNotificacao;
 end;

var
 FrMain: TFrMain;

implementation

{$R *.dfm}

procedure TFrMain.BtnNotificarClick(Sender: TObject);
begin
 // "Pedaço" de código que contém a lógica de notificação
 // é injetado em Notificacao e chamado internamente,
 // porém passado externamente, como método anônimo
  Aviso := Notificacao.Create(
    procedure (Texto: string)
    begin
      ShowMessage(Texto + ' notificado via ' + rdForma.Items[rdForma.ItemIndex]);
    end
  );
  try
    // Send internamente vai chamar o bloco de código acima
    Aviso.Send(edtTexto.Text);
  finally
    Aviso.Free();
  end;
end;

end.
```

tudo é feito de forma mais fluente, injetando diretamente a funcionalidade, sem um nome.

Futuro: Expressões Lambda no Delphi

E finalmente, vou mostrar um recurso que existe já no Delphi Prism e no C#, e que provavelmente poderemos enxergar em futuras implementações do compilador do Delphi Win32 / 64, são as Lambda Expressions. É simplesmente uma forma mais enxuta de escrever um método anônimo, isso quer dizer que no futuro poderemos escrever este mesmo exemplo assim (esta é uma previsão):

```
Aviso := Notificacao.Create(Texto -> ShowMessage(Texto));
```

Conclusão

A palavra-chave é criar sistemas mais flexíveis, de baixo acoplamento, mas para isso, vimos que é necessário um estudo profundo das boas práticas e dos recursos que a linguagem e do IDE nos oferece para aplicá-los. Vimos que uma Boa Prática não deve levar só em consideração aspectos como legibilidade e design, mas também performance, segurança e outros requisitos. Pensar em um bom design de software, prevenir "bad smells", escrever código elegante e refatoração constante, dominando e usufruindo o que há de melhor no Delphi, foram os pontos fortes aqui discutidos. Bom proveito e bom código.

Guinther Pauli
http://gpauli.com
http://twitter.com/guintherpauli
http://www.facebook.com/guinther.pauli

Consultor independente, Professor de Pós-Graduação, Membro da Comissão Editorial das Revistas .NET Magazine Brasil e ClubeDelphi, Certificado oficial Microsoft MCP, MCAD, MCSD.NET, MCTS, MCPD (Arquitetura, ASP.NET e C# - 4.0) - Certificado oficial Delphi: 3, 5, 6, 7, 2005, 2006, Delphi para Web. Foi arquiteto e desenvolvedor do sistema de doações on-line do projeto Criança Esperança da Rede Globo / Unesco. Beta Test de várias versões do Delphi, palestrante em 5 edições da Borland Conference, maior evento de desenvolvedores do país. Ministra treinamentos corporativos para empresas e universidades, já tendo treinado mais de 5 mil profissionais em todo Brasil e exterior. Publicou mais de 120 artigos e 700 video aulas, além do livro "Delphi – Programação para Banco de Dados e Web". Bacharel em Sistemas de Informação e Mestrando em Informática na UFSM, com ênfase em Best Practices.

Blogs do Autor
http://gpauli.com
http://twitter.com/guintherpauli
http://www.facebook.com/guinther.pauli

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Refatoração Fácil

Conheça e aplique

Resumo DevMan

De que trata o artigo:

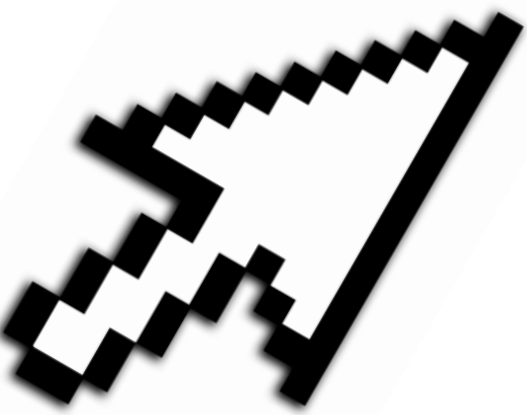
Este artigo tem por fim efetuar uma introdução ao conceito de refatoração, demonstrando suas principais características, vantagens e diversas técnicas que quando utilizadas em conjunto, permitem um ganho de produtividade em eventuais manutenções do código fonte, apresentando também os recursos que o Delphi XE 2 oferece para aplicar este processo.

Em que situação o tema é útil:

Aplicar a Refatoração em Softwares que sofrem constantes alterações para que suas futuras manutenções possam ser efetuadas de maneira mais rápida e intuitiva através da utilização de diversas técnicas, boas práticas e ferramentas que permitem que este processo possa ser facilitado e automatizado ao máximo.

Refatoração:

A manutenção de um software é um processo muitas vezes inevitável e inadiável onde o sistema em questão recebe diversas alterações que não estavam previstas na concepção do mesmo. São códigos e mais códigos que são utilizados para adaptar o software de acordo com a nova necessidade. Nesse movimento todo, trechos de códigos são alterados, movidos ou duplicados e por isso podem tornar o código fonte em geral mais difícil de ser intepretado e trabalhado pela equipe. Durante a vida de um Software, convém analisar se o mesmo necessita do processo de Refatoração. Este processo consiste em um conjunto de diversas técnicas e boas práticas que visam reduzir o código fonte, tornar o mesmo mais fácil de ser interpretado, diminuir a possibilidade de erros, reduzir o tempo gasto para eventuais manutenções, dentre outras vantagens, sem que os códigos fontes tenham seus propósitos finais alterados, ou em outras palavras, chegar ao mesmo resultado de uma maneira mais fácil, intuitiva e padronizada. Neste artigo primeiramente é abordada uma introdução ao conceito de Refatoração, bem como algumas de suas técnicas mais comuns que podem ser aplicadas em diversas linguagens. Posteriormente, são demonstrados os recursos que o Delphi XE 2 oferece para facilitar e automatizar a aplicação deste processo que pode aumentar muito a produtividade. Muitos destes recursos também estão presentes em outras versões do Delphi.



O desenvolvimento de um Software é um projeto que engloba um planejamento, diversas fases, ferramentas e técnicas. Entretanto, desde sua concepção, mesmo um software que tenha sido extremamente bem planejado e organizado, pode vir a sofrer diversas mudanças "inesperadas" em sua estrutura para que o mesmo possa ser adaptado às novas necessidades do dia-a-dia, uma vez que Softwares integrados não dependem apenas da entidade em si, mas também de fatores externos como é o caso da apresentação de documentos fiscais perante a Receita, entre outros vários fatores. Sendo assim, muitas vezes o projeto precisa ser revisto e alterado de última hora para atender aos novos requisitos e seu código fonte pode sofrer diversas alterações que não foram devidamente previstas anteriormente, levando em consideração também que pode haver outro desenvolvedor destinado a esta tarefa, aplicando assim sua própria forma de desenvolver e não seguindo um padrão pré-definido anteriormente pelos outros desenvolvedores do sistema.

GIULIANO SCOMBATTI PINTO

A Refatoração ou Refactoring é um conceito constituído de diversas técnicas que consistem basicamente em alterar o código fonte de um Software existente para que o mesmo se torne mais confiável, simples, flexível, intuitivo e sustentável, sem que sua funcionalidade possa ser afetada, melhorando também a estrutura interna do Software em questão. Com uma maior facilidade na interpretação e alteração do código, a manutenção ou o desenvolvimento realizado pela equipe se torna muito mais ágil, minimizando a quantidade de erros e consequentemente aumentando significativamente a qualidade do mesmo. As técnicas provenientes da Refatoração também são constituídas de diversas boas práticas de programação.

O processo de Refatoração poderia ser resumido em uma "revisão" e organização do código e é divido em diversas fases simples e pequenas, mas que quando são utilizadas em conjunto permitem um efeito bastante significativo. Evidentemente, toda alteração que é realizada no código fonte, como é o caso da aplicação das técnicas de Refatoração, por menor que possam ser, implica em uma nova fase de testes onde o ideal é que a cada alteração ou fase que seja concretizada, sejam efetuados diversos testes em seguida para facilitar na identificação de possíveis erros, uma vez que não convém efetuar diversas técnicas de Refatoração de uma vez e só então efetuar os testes, pois seria muito mais difícil encontrar possíveis falhas.

Além das vantagens obtidas ao utilizar a Refotoração já comentadas, normalmente também é possível obter uma redução do código fonte do software, uma vez que o código pode conter métodos duplicados ou funções desnecessárias.

Com relação ao momento certo ou necessário para aplicar o processo de Refatoração, não existe uma determinada regra que deve ser seguida ou uma obrigatoriedade na sua implementação. Contudo, este processo é muito indicado sempre que existir a necessidade de adicionar uma nova funcionalidade ao software, corrigir um erro, na reutilização de um código para outro sistema ou durante uma revisão geral do código fonte, para uma melhor compreensão. De fato o processo de Refatoração leva tempo para ser aplicado, porém seu resultado é conseguido em longo prazo, implicando em uma redução do tempo para as futuras alterações ou manutenções do software.

Como pode ser observado a Refatoração não é uma técnica exclusiva de uma determinada linguagem, mas sim conceitos que podem ser aplicados nos códigos fontes dos Softwares nas mais variadas ferramentas de desenvolvimento presentes no mercado, entretanto, ao conhecer tais técnicas utilizadas para este processo convém já aplicá-las nos novos projetos para minimizar ao máximo futuras alterações complexas ou revisões de todo o código fonte. Vamos conhecer algumas destas técnicas.

Extração de método

Em diversos momentos no desenvolvimento de um Software, um determinado trecho de código pode ser reaproveitado. Entretanto, às vezes este código já se encontra dentro de um método e muitas vezes ele é copiado para um novo método criado, onde por sua vez o método original é mantido intacto, gerando assim certa duplicidade de código.

O processo de extração de método é indicado sempre que ocorrer esta situação, para que não exista código fonte duplicado na aplicação, sendo uma das técnicas mais utilizadas na Refatoração. Além disso, ela também deve ser aplicada em casos onde o código fonte de determinado método é muito extenso e o mesmo necessita ter vários comentários internos para que possa ser compreendido facilmente. Sendo assim, são retiradas partes destes códigos do método principal, originando novos métodos, onde posteriormente são efetuadas chamadas para tais métodos criados no método original, que teve seu código fonte removido. Se este processo for analisado, métodos menores são mais fáceis de serem compreendidos. Em contrapartida ao dividir um método maior em outros menores, a única desvantagem que se pode ter, dependendo da alteração que tenha que ser realizada, é o fato de ter que percorrer os outros métodos que são chamados para efetuar a correção. Evidentemente a aplicação desta boa prática sempre vai depender da necessidade e viabilidade do projeto em questão. Contudo, esta técnica permite que o método principal tenha seu código minimizado devido à chamada aos novos métodos, mas ele continua com o mesmo propósito, bem como tais novos métodos podem ser reutilizados em outras partes da aplicação, facilitando também futuras manutenções necessárias, uma vez que ocorre uma centralização das funções. Como pôde ser observado, a extração de método pode ser resumida como a divisão de um método maior que executa diversas tarefas em métodos menores que podem ser reaproveitados. Contudo, outro ponto que deve ser observado ao adotar esta técnica é com relação aos nomes dos métodos, que devem descrever a função dos mesmos e não a maneira com que eles fazem seus objetivos. Também vale a pena ressaltar que devem ser definidos os parâmetros que serão necessários para a utilização do novo método, onde uma boa prática é passar todas as variáveis locais que pertencem ao método original para o novo através de tais parâmetros.

Extração de classe

Outra técnica mais complexa que a apresentada anteriormente e que também faz parte do processo de Refatoração é a extração de classe. Ela deve ser aplicada quando uma grande quantidade de código fonte é centralizado em determinada classe e a mesma se encontra "sobrecarregada", ou em outras palavras, a classe é responsável por diversas tarefas sendo que a mesma poderia ser divida em duas ou mais classes. Este é um fato muito comum, pois durante as alterações mais e mais códigos são adicionados para atender as novas necessidades e no final das contas, uma classe pode acabar contendo vários métodos que até dificultam o entendimento e futuras manutenções. Sendo assim, ao aplicar esta técnica a classe sobrecarregada é dividida em outras classes e evidentemente precisam ser criadas as referências necessárias entre ambas para que as funcionalidades mantenham-se intactas.

Classe alinhada

Em contrapartida, outra técnica denominada classe alinhada tem uma função oposta à técnica de extração de classe. Classe alinhada deve ser adotada quando determinada classe tem poucas tarefas e na verdade deveria ter as funções que estão presentes em outras classes, sendo assim, esta técnica implica que os objetos e métodos devem ser movidos para a mesma. Este é um caso de Refatoração inversa que pode ocorrer em diversos momentos durante o processo, implicando que cada caso deve ser devidamente estudado para a execução da técnica adequada para aquela situação.

Relocação de método

Outra técnica importante neste contexto é a de mover um método. Esta técnica normalmente é muito utilizada por outras técnicas de Refatoração que são mais complexas. Como seu próprio nome sugere, ela é utilizada para mover determinado método presente em uma classe para outra classe, e às vezes alguns ajustes deverão ser efetuados. Contudo, também devem ser observados quais são os objetos do método que será movido que pertencem à classe antiga que também deverão ser movidos para a nova classe, bem como verificar a existência de outros métodos relacionados com o mesmo que necessitem ser movidos.

Variáveis em linha e renomeação

Outra técnica utilizada é a de variáveis em linha, também conhecida como Inline Temp. Ela é aplicada quando determinada variável local possa interferir em outra técnica de Refatoração, onde tal variável pode ser substituída por uma simples expressão. Contudo, às vezes tal variável temporária pode ser mantida sem problemas caso o desenvolvedor necessite utilizá-la para outros fins em seu código fonte.

Dando continuidade, mais uma técnica importante é a de renomear os nomes dos métodos em geral, que também não deve ser ignorada no processo de Refatoração. Como sugerido pela sua própria denominação, ela consiste em renomear o método em questão, pois muitas vezes depois de várias manutenções os métodos podem ter suas funcionalidades estendidas e acabam efetuando outras tarefas que diferem do nome original, sendo assim o mesmo deve ser revisto e ter seu nome alterado para se referir ou descrever fielmente seu objetivo.

Encapsulamento de campos

Esta técnica tem por objetivo adicionar propriedades a determinado objeto, ou seja, substituir determinado campo público por um campo privado, onde no Delphi é seguida uma padronização que identifica esses campos, normalmente estes iniciam com a letra "F". Esta técnica permite um maior controle do campo, uma vez que o mesmo agora é acessado por uma propriedade (property) ou ainda por métodos de leitura (get) e gravação (set).

Outras diversas

Dando sequência, outra técnica consiste na substituição de variáveis locais utilizadas como sinalizadores para o controle do momento em que determinado loop deve deixar de ser executado, por comandos como Exit, Break e Continue.

A substituição de métodos extensos por métodos de objetos também é outra técnica indicada no processo de Refatoração. Ela pode ser aplicada sempre que um determinado método extenso, constituído de várias variáveis, tenha que ser dividido em menores, porém não convém utilizar a técnica de extração de método. Neste caso, tal método pode ser convertido para uma classe contendo diversos métodos menores, onde as variáveis locais deste método convertido devem se tornar propriedades da mesma.

A técnica de Ocultação/Delegação também é outro processo que deve ser aplicado. Em suma, ela consiste na ocultação de classes inteiras e seus campos encapsulados. O padrão Facade pode ser uma boa aplicação para esta técnica.

Nota do DevMan

O padrão Facade ou Façade (Fachada) tem como objetivo principal apresentar uma única interface para o conjunto de outras interfaces presentes no subsistema, fazendo com que tal subsistema possa ser mais simples de ser utilizado, uma vez que reduz a complexidade na utilização dos recursos internos que podem existir nas classes em questão. Este padrão também é baseado na aplicação de diversas técnicas da Orientação a Objetos, como o Encapsulamento.

Refatoração no Delphi XE 2

O Delphi traz consigo um menu denominado Refactor que contém diversas opções para aplicar técnicas de Refatoração. O desenvolvedor também pode posicionar o cursor do mouse na área do código onde deseja aplicar uma Refatoração ou selecionar determinado trecho de código e posteriormente clicar com o botão direito (menu de contexto) e acessar as opções do menu Refactor de acordo com a área selecionada. Feito este processo, é aberta uma janela denominada Refactorings onde o desenvolvedor pode aplicar a Refatoração ou cancelar a mesma, contando também com a opção de desfazer, caso houver necessidade.

A **Figura 1** demonstra o menu Refactor presente no menu de contexto ou pop-up do Delphi XE 2 de acordo com a seleção atual que é uma variável que foi denominada Soma.

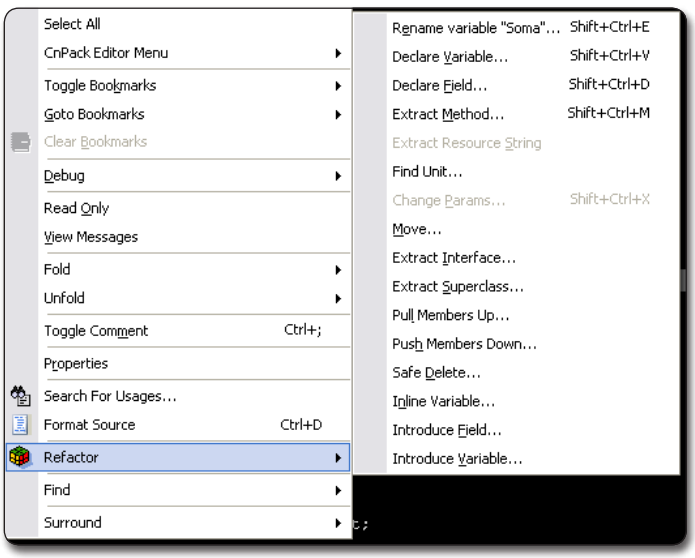


Figura 1. Menu Refactor

Denominção das técnicas de refatoração do Delphi XE 2 e funções

Rename

Sempre que um método possui seu código fonte alterado, muitas vezes o mesmo pode ter sua função ampliada ou minimizada, o que implica que o nome antigo do mesmo já não descreve fielmente o novo propósito que foi estabelecido a ele. Uma vez que este método pode ser chamado por diversas Units no projeto Delphi, o desenvolvedor poderia ter um trabalho extra se tivesse que buscar e renomear cada chamada que é efetuada. Sendo assim, a refatoração Rename (Renomear), que pode ser acessada através do menu de contexto Refactor>Rename, permite que um método seja renomeado de uma maneira bem prática e automática, onde o próprio Delphi pode buscar a declaração do mesmo, bem como todas as chamadas que são realizadas ao método em questão e permitir que o desenvolvedor renomeie e aplique esta Refatoração. Entretanto, este processo não é destinado apenas a métodos, mas também pode ser aplicado em variáveis, propriedades, campos, etc, cuja função é a mesma, renomear todas as ocorrências encontradas no projeto. É importante lembrar que para que o procedimento de Rename funcione corretamente, deve-se selecionar o método e só então informar o novo nome. Deste modo, tanto a declaração, quanto o corpo do método serão alterados.

Declare Variable

Um recurso muito útil e automatizado é a técnica de declaração de variável (Declare Variable). Durante a inserção ou alteração de um determinado trecho de código fonte, pode surgir a necessidade de utilizar uma variável no escopo atual que não foi previamente declarada. Sendo assim, o desenvolvedor não necessita retornar até a seção de declaração de variáveis (var) para declarar a mesma manualmente. Para tanto, basta utilizar a Refatoração Declare Variable, presente no menu de contexto Refactor>Declare Variable, para que o Delphi se encarregue de efetuar a declaração da variável automaticamente. Quando esta técnica é utilizada, uma janela é aberta para que os detalhes desta nova variável possam ser definidos, tal como o tipo da mesma. Supondo que em determinado trecho do código o desenvolvedor necessite utilizar uma nova variável, que neste exemplo foi denominada "Soma", ele simplesmente pode digitá-la e em seguida clicar com o botão direito sobre a mesma, acessar o menu Refactor e selecionar Declare Variable, definir o tipo da variável e suas opções para que o Delphi se encarregue de adicioná-la automaticamente na seção var. A **Figura 2** demonstra a janela Declare Variable para a nova variável "Soma".

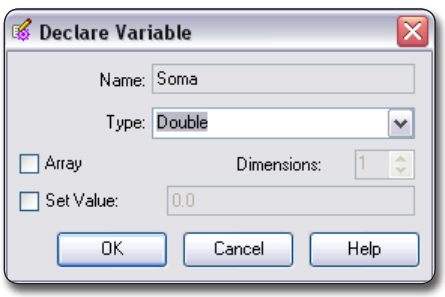


Figura 2. Janela da refatoração Declare Variable

Declare Field

De maneira semelhante a Declare Variable, a Refatoração Declaração de Campo, (Declare Field), presente no menu de contexto Refactor>Declare Field, pode ser utilizada para automatizar a declaração de um campo necessário que não foi previamente declarado, onde o desenvolvedor não necessita retornar a seção de declarações para informar o mesmo de maneira manual. Ao utilizar esta técnica, uma janela é aberta para que as definições do campo possam ser efetuadas, tal como a visibilidade do mesmo (private, public, protected, etc). Supondo que o desenvolvedor necessite criar um campo no escopo atual do código fonte, que neste exemplo foi denominado "Soma", ele pode digitá-lo e em seguida clicar com o botão direito sobre o mesmo, acessar o menu Refactor>Declare Field, definir o tipo do campo e sua visibilidade para que o Delphi o declare automaticamente. A **Figura 3** demostra a janela Declare Field para criar o novo campo "Soma".

Extract Method

A Refatoração Extração de Método (Extract Method), uma técnica que foi comentada anteriormente com uma visão geral, tem a função de extrair determinado código fonte selecionado, para um novo método, que por sua vez receberá uma lista de parâmetros e declarações de variáveis automaticamente, e na linha atual onde o código se encontrava é gerada uma chamada para o mesmo de maneira automática pelo Delphi. Ao utilizar esta técnica, uma simples janela é aberta para que o nome do método seja informado.

Basta para isso selecionar o trecho de código e a opção Extract Method do menu Refactor.

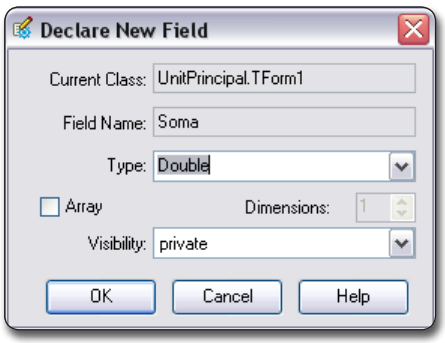


Figura 3. Janela da refatoração Declare Field

Extract Resource String

Normalmente quando os valores de variáveis do tipo String são atribuídos diretamente no código, podem deixar o trecho atual de código fonte mais extenso e em casos onde são inseridos vários caracteres até "poluir o código". O Delphi conta com a técnica de refatoração Extract Resource String que cria automaticamente uma seção denominada resourcestring, que tem seu funcionamento semelhante ao de constantes (const), onde é declarado um elemento e seu devido valor em caracteres. A utilização de deste recurso é muito útil, pois são várias as situações onde determinada cadeia de caracteres pode ser reutilizada, tais como em mensagens e comandos SQL. Para utilizar esta técnica, basta posicionar o cursor do mouse sobre o trecho de código que contém a String, posteriormente abrir o menu de contexto clicando com o botão direito do mouse e acessar "Extract Resource String", presente no menu Refactor. Supondo que o desenvolvedor necessite criar uma mensagem apenas para indicar que um processo foi concluído com êxito, ele poderia utilizar esta técnica de Refatoração, extraindo assim a mensagem e permitindo a reutilização da mesma futuramente através do menu citado. Um exemplo da utilização desta técnica é apresentado a seguir:

```
resourcestring
  StrProcessoConcluido = 'O processo foi concluído com êxito!';
...
  ShowMessage(StrProcessoConcluido);
```

Com isto a Resourcestring denominada StrProcessoConcluido pode ser reutilizada quantas vezes houver a necessidade, facilitando o trabalho do desenvolvedor.

Find Unit

No desenvolvimento de um Software muitas vezes existe a necessidade de utilizar na Unit atual funções, procedimentos, objetos, classes e tipos que estão presentes em outras Units. Sendo assim, a refatoração Find Unit (Localizar Unit), que pode ser acessada através do menu de contexto Refactor>Find Unit, permite que a origem (Unit) de tais elementos possa ser localizada facilmente e declarada automaticamente na classe atual na seção Interface ou Implementation, facilitando a localização e utilização de elementos compartilhados. A **Figura 4** demonstra a janela da refatoração Find Unit. É possível também acessar esta janela por meio do atalho Ctrl+Shift +A. Bastando que ao digitar o nome de um método em Search, os possíveis resultados sejam exibidos.

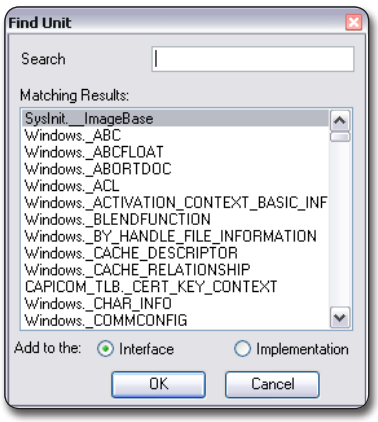


Figura 4. Janela da refatoração Find Unit

Change Params

É comum se deparar com situações onde determinado método necessite de novos parâmetros. A Refatoração Change Params (Alterar Parâmetros) é um recurso muito útil destinado a inserção, alteração ou exclusão dos parâmetros referentes a determinado método selecionado, onde o Delphi se encarrega de efetuar as atualizações automaticamente. Quando esta técnica é utilizada, uma janela é aberta contendo diversas alternativas para que o desenvolvedor possa editar tais parâmetros sem que o mesmo tenha que ir até a declaração do método para efetuar este processo. Supondo que o desenvolvedor deseja alterar os parâmetros de determinada função, que neste exemplo foi chamada de "VerificarStatus()", basta posicionar o cursor do mouse sobre a mesma e através do menu de contexto, selecionar o menu Refactor>Change Params. A **Figura 5** apresenta a janela da Refatoração Change Params para o procedimento de exemplo VerificarStatus().

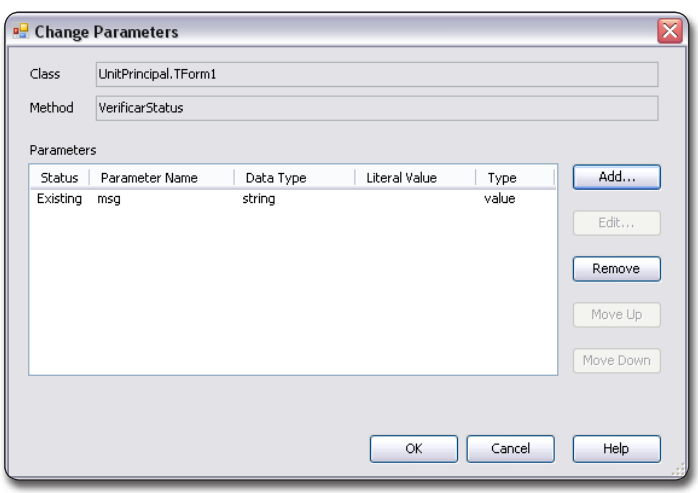


Figura 5. Janela da refatoração Change Params

Extract Interface

Extract Interface, como sugerido pelo seu próprio nome, é utilizada para a extração de interfaces. Contudo, para que uma interface possa ser extraída existe um fator que deve ser levado em consideração, o fato de que a extração só é permitida para métodos não estáticos. Ao ser extraída a Interface, são criados métodos públicos na mesma e se já existir uma Interface com o mesmo nome que foi especificado no momento da extração, os métodos serão extraídos para a Interface já existente. Para utilizar este recurso, basta posicionar o mouse sobre o trecho do código fonte desejado e através do menu de contexto, selecionar a opção Refactor>Extract Interface.

Extract Superclass

A Extract Superclass é utilizada para a extração de vários elementos, como métodos, eventos, propriedades, campos, entre outros, para uma classe que é automaticamente criada pelo Delphi junto com tais elementos que devem ser selecionados. Ao utilizar esta técnica, o desenvolvedor informa qual será o nome da classe, bem como a Unit em que a mesma será criada, entre outras opções. Para utilizar este recurso, basta posicionar o mouse sobre o trecho do código fonte desejado e através do menu de contexto, selecionar a opção Refactor>Extract Superclass.

Pull members up e Push members down

As refatorações Pull member sup e Push members down são utilizadas para mover os membros (objetos, métodos, etc) para outra classe presente na hierarquia, de maneira ascendente ou descendente (classe base ou derivada). Esta técnica também pode ser aplicada através da janela ModelView (View>ModelView), que permite uma melhor visualização dos elementos da classe, e posteriormente também através do menu de contexto, acessando Refactor>Pull members up ou Push members down. Isso é extremamente útil, pois permite uma facilidade ao deslocar os membros entre as classes mais baixas. Supondo para a situação o seguinte trecho de código:

```
type
 TCustomPessoa = class
 public
 end;

 TPessoa = class(TCustomPessoa)
 private
  FNome: String;
  procedure SetNome(const Value: String);
 public
  property Nome : String read FNome write SetNome;
 end;
```

Na situação da listagem anterior, há uma hierarquia entre a classe TPessoa e TCustomPessoa, na qual TPessoa é filha de TCustomPessoa. Supondo que o objetivo seja deslocar o Field Nome para TCustomPessoa. Deve-se selecionar a propriedade e com o clique de direita acionar a opção Refactor>Pull members Up. Note pela janela da **Figura 6** que uma tela é exibida sugerindo o que deve ser levado para o método anterior.Como a propriedade utiliza o campo FNome para leitura (get) e a procedure SetNome para gravação (método set) , o próprio Wizard ilustra estas alterações para que seja as dependências sejam deslocadas juntamente.

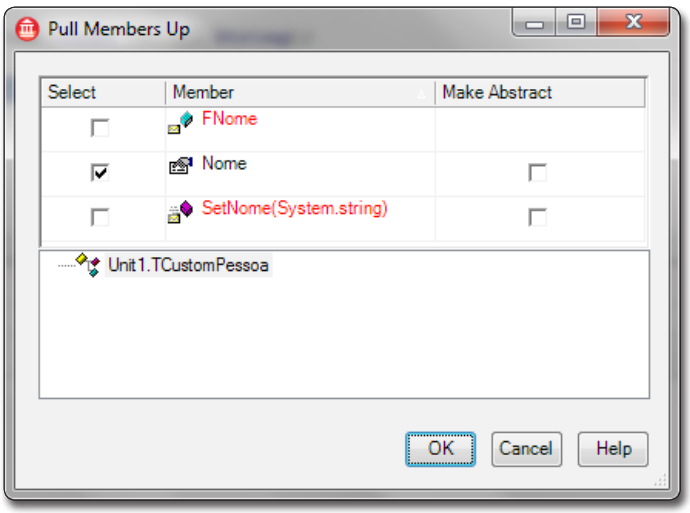


Figura 6. Janela de Pull Members Up

Já no próximo passo (**Figura 7**) é ilustrado o efeito das alterações por meio da própria janela de Refatoração. É possível aceitar ou não as alterações. A seguir as modificações realizadas a partir do aplicação do Refactoring:

```
TCustomPessoa = class
public
 private
  FNome: string;
  procedure SetNome(const Value: string);
 public
  property Nome: string read FNome write SetNome;
 end;

 TPessoa = class(TCustomPessoa)
end;
```

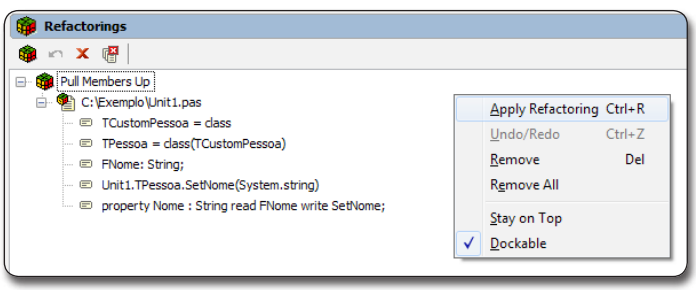


Figura 7. Demonstrando as alterações do Pull Members Up

Safe Delete

A refatoração Safe Delete também é um recurso interessante, pois, permite que determinado objeto ou método seja apagado de maneira segura, ou seja, verifica se o mesmo não é utilizado por outros métodos para que possa ser removido, e caso seja, também identifica o número de vezes que o mesmo é chamado,

permitindo também uma visualização dos mesmos. Para utilizar este recurso, basta posicionar o mouse sobre o método ou objeto e através do menu de contexto, selecionar Refactor>Safe Delete A **Figura 8** demonstra a janela que é exibida contendo as informações para a aplicação da Refatoração Safe Delete, onde neste exemplo o procedimento VerificarStatus() foi utilizado como base e consequentemente ele não pode ser deletado seguramente pois contém uma chamada.

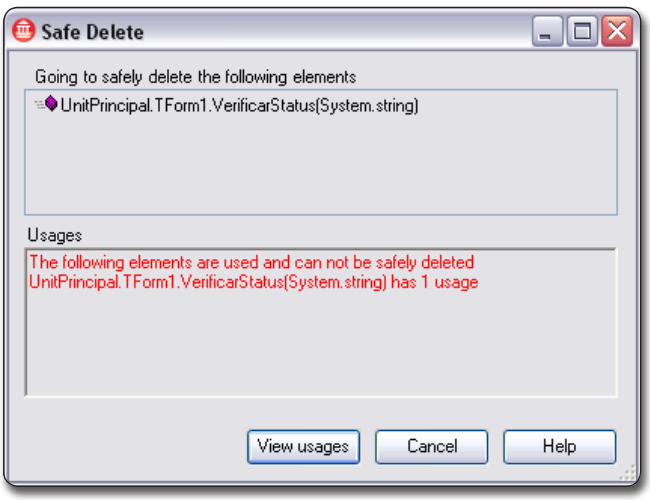


Figura 8. Janela da refatoração Safe Delete

Introduce Variable

Introduce Variable (Introduzir Variável) é destinada a criação automática de uma variável local, bem como a definição de seu tipo, para armazenar uma determinada expressão selecionada ou valor. Ao utilizar este recurso uma janela é aberta solicitando o nome para a nova variável. Um exemplo da utilização desta Refatoração pode ser observado a seguir:

```
Resultado := Soma - Resto;
```

Após a aplicação da Refatorção Introduce Variable, o resultado obtido será:

```
Conta := Soma - Resto;
Resultado := Conta;
```

Neste exemplo, o Delphi se encarregou de criar uma nova variável que foi denominada Conta do tipo Double para conter a expressão que foi apresentada. Este resultado pode ser obtido através da seleção da expressão "Soma – Resto" e posteriormente com a utilização do menu de contexto Refactor>Introduce Variable.

Inline Variable

A Refatoração Inline Variable tem um efeito oposto ao recurso Introduce Variable, em outras palavras, todos os locais onde a variável selecionada está presente são substituídos pela expressão ou valor que ela contém. Este é mais um exemplo de Refatoração inversa que pode ser aplicado através do menu de contexto Refactor>Inline Variable.

Conclusão

A Refatoração é um processo opcional e que de início pode causar certo receio por parte dos desenvolvedores em aplicá-lo, pois propõe diversas alterações no código fonte para que o mesmo se torne mais legível e intuitivo, o que obviamente necessitará de tempo, porém, este tempo utilizado é compensado no futuro, uma vez que a Refatoração quando devidamente aplicada, permite um ganho de produtividade em longo prazo. Entretanto, várias ferramentas atuais de desenvolvimento, como é o caso do Delphi, disponibilizam uma série de recursos que automatizam ao máximo a aplicação de Refatoração no código fonte, permitindo assim que este processo também possa ser feito de uma maneira mais rápida.

Como pôde ser observado a Refatoração conta com diversas boas práticas de programação e várias técnicas. Sendo assim, por ter boa parte constituída por conceitos, ao desenvolver um novo projeto tais boas práticas já podem ser adotadas por parte da equipe de desenvolvimento, visando assim à concepção de códigos fontes que futuramente necessitem o mínimo possível de Refatorações, diminuindo assim o tempo que eventualmente seria necessário em alterações.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Informações sobre conceitos e técnicas de Refatoração
http://martinfowler.com/refactoring/

DevSpace do Autor
http://www.devmedia.com.br/space.asp?id=283274

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

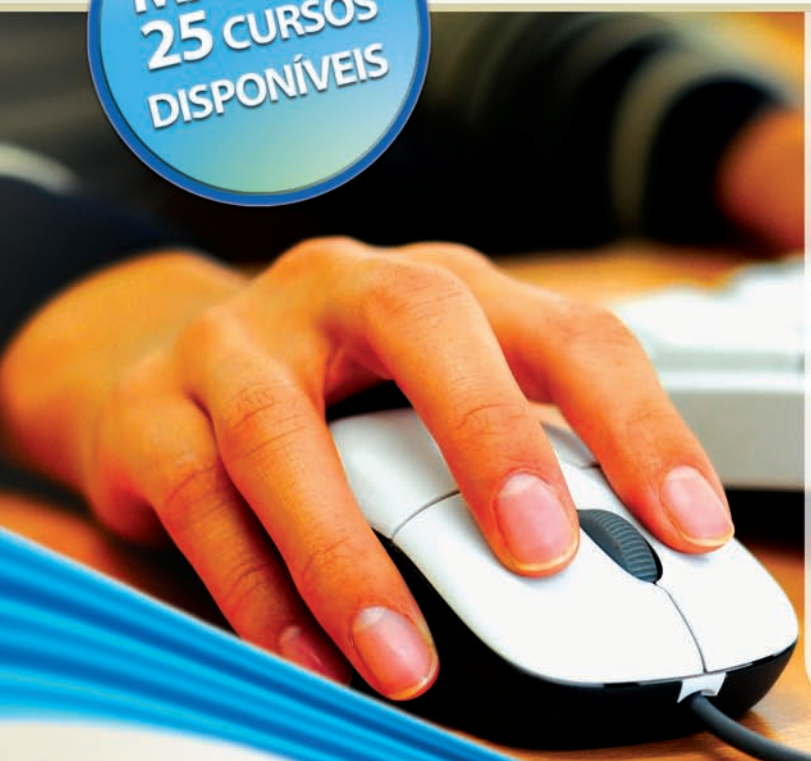
www.devmedia.com.br/clubedelphi/feedback

CURSOS ONLINE

.net magazine

A Revista .net Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

MAIS de 25 CURSOS DISPONÍVEIS



CONHEÇA OS CURSOS MAIS RECENTES:

- Curso .net Entity Framework - Loja Virtual
- Novidades do Visual Studio 2010
- Componentes DevExpress
- Crie um chat utilizando Asp.Net 3.5 e Linq
- Novidades do C# (csharp) 3.0
- POO com C#
- Microsoft Visual Studio Tools for Office (VSTO)
- Computação na Nuvem com a Plataforma Microsoft Windows Azure

Para mais informações :

www.devmedia.com.br/curso/netmagazine

(21) 3382-5038

http://www.devmedia.com.br

DevMedia

DevMedia

DÊ UM SALTO EM CONHECIMENTO!

Acesse o maior portal para desenvolvedores da América Latina!

17 mil posts

280 mil cadastrados

5 milhões de page-views por mês

http://www.devmedia.com.br/

DevMedia